PDF - Fassung

Norbert Schwarz

Einführung in TEX

“unveränderte”
PDF-Fassung der
3. Auflage von 1991

Vorwort zur PDF-Ausgabe

Das vorliegende PDF-Dokument ist die elektronische Ausgabe des Buches

"Einführung in TEX"

das in der 3. Auflage im Jahr 1988 im Addison-Wesley Verlag Deutschland erschien.

Der Vertrieb ist durch den Verlag eingestellt und die Rechte sind an mich zurückgegeben worden.

Die PDF-Fassung ist eine adaptierte, inhaltlich unveränderte Fassung des gedruckten Exemplars. Es sind lediglich die Möglichkeiten, Verweise in PDF-Dokumenten in Hyper-Link-Form zu nutzen, im Inhaltsverzeichnis und Anhang genutzt worden.

Ich habe mich entschieden, in dieser elektronischen Form die Zugänglichkeit zu diesem Werk aufrechtzuerhalten, und hoffe, dass dies dem einen oder anderen noch nutzt.

Für den persönlichen Bedarf mag sich jeder auch gedruckte Exemplare herstellen. Vertriebsrechte für gedruckte, zu verkaufende Exemplare behalte ich mir vor.

Norbert Schwarz

Februar 2002

Email: `Norbert.Schwarz@ruhr-uni-bochum.de`

Vorwort

Seit mehreren Jahren beschäftige ich mich mit dem TEX-System, zum einen als Anwender bei eigenen Dokumenten, zum anderen in der Ausbildung von Studenten. Zur Unterrichtsunterstützung entstand zunächst ein deutschsprachiges Skript, aus dem sich dann dieses Buch entwickelt hat.

Die Aufgabe dieses Buches ist kein Ersatz von Donald E. Knuth's "The TEXbook", dem ich hier für seine Arbeit meine Reverenz erweisen möchte; eine Reihe von Beispielen und Anwendungsfällen haben ihren Ursprung in seinem Referenzwerk.

Hier soll eine *Einführung* in die Benutzung des TEX-Systems gegeben werden, auch mit Hinweisen im Hinblick auf Unhandlichkeiten, die sich aus einem international anwendbaren Programm im Spezialfall der deutschen Sprache ergeben.

Auch möchte ich keinen Anspruch auf eine vollständige Darstellung aller TEX-Parameter und Steuerbefehle erheben. Selbst heute stoße ich noch gelegentlich auf neue Anwendungsmöglichkeiten. TEX kennt etwa 900 Kommandos, davon sind gut 300 fest programmierte Leistungen; die restlichen Befehle sind die Makros des *'plain-TEX'*, das im allgemeinen mit *'TEX'* identifiziert wird.

Dies soll den Leser aber nicht abschrecken; denn es wird nur ein sehr geringer Teil dieses Befehlsschatzes ständig benötigt. Der größte Reiz des TEX-Systems liegt in seinem Entwicklungspotential. Es bietet in seiner Naturform schon eine leistungsfähige Umgebung, diese kann sich jeder nach seinen Wünschen umgestalten und erweitern.

Vorwort zur 2. Auflage

In der Neuauflage wurden einige Fehler beseitigt. Einige Abschnitte sind geringfügig verändert worden. Insbesondere ist eine Darstellung der `\read`- und `\write`-Befehle zur Registerbildung hinzugefügt worden. Die Kurzbeschreibung der TEX-Befehle ist jetzt ergänzt durch eine Kennzeichnung, welche TEX-Befehle fest einprogrammiert sind.

Vorwort zur 3. Auflage

Die Änderungen in der 3. Auflage sind bedingt durch die Leistungserweiterungen in TEX selbst ab der Version 3.0. Hier sind insbesondere für den deutschsprachigen Raum für den Anwender wichtige zusätzliche Möglichkeiten hinzugekommen, wie die direkte Eingabe von Umlauten aufgrund der neuen Verarbeitungsfähigkeit von 256 Zeichen umfassenden Zeichensätze, Mehrsprachigkeit beim Trennen und die Beeinflussung des Absatzumbruches. Neu hinzugekommen sind die folgenden Befehle:

`\badness`, `\emergencystretch`, `\errorcontextlines`, `\holdinserts`, `\inputlineno`, `\language`, `\setlanguage`, `\newlanguage`, `\lefthyphenmin`, `\righthyphenmin`, `\noboundary`, `\topglue`.

Daneben habe ich eine Reihe von Anregungen aufgegriffen, um einzelne Bereiche ausführlicher darzustellen. Beispielsweise ist die Kurzbeschreibung aller plain-TEX-Befehle stark erweitert worden.

Noch einige allgemeine Anmerkungen:

Die Gruppe der TEX- und LaTEX-Anwender ist in sprunghaftem Wachstum begriffen, insbesondere seitdem das Programm den universitär informatischen Elfenbeinturm verlassen hat, und die Probleme von der Implementierung zur Benutzung verlagert sind. So möchte ich hier auch auf die Existenz einer deutschsprachigen TEX-Anwendervereinigung hinweisen, die ihren Mitgliedern Unterstützung bei der TEX-Anwendung bietet, gerade auch durch Bereitstellung aktueller Softwareversionen.

Die Anschrift für *schriftliche* Anfragen lautet:

DANTE
Deutschsprachige Anwendervereinigung TEX e. V.
c/o Rechenzentrum der Universität Heidelberg
Im Neuenheimer Feld 293
D–6900 Heidelberg 1

Norbert Schwarz, Bochum im März 1991

Inhalt

1 Allgemeines

1 Charakteristika des TEX-Systems 11

2 Bedienung

1 Befehlsstruktur und Befehlszeichen 13
2 Anwendung der Befehle 15
3 Satzzeichen 16
4 Motivierendes Beispiel 17
5 Programmstart und Programmende 20

3 Textsatz

1 Das normale Druckbild 21
2 Maßeinheiten 22
3 Umlaute und nationale Sonderzeichen mit Version 2 24
4 Umlaute und Akzente mit Version 3 26
5 Abstände zwischen Absätzen 27
6 Abstände zwischen Wörtern 28
7 Zeilenabstand 30
8 Zeilenausrichtung 33
9 Absatzausrichtung 36
10 Vielfache Absatzformen 39
11 Schmale Absätze und Umbruchsteuerung 40
12 Einrückungen und Listen 42
13 Seitenwechsel 45
14 Seitennumerierung 46
15 Kolumnentitel: Seitenüberschriften, Seitenunterschriften 47
16 Einfügen von Illustrationen 49
17 Fußnoten 50
18 Trennen 51
19 Waagerechte und senkrechte Striche 53
20 Fehlermarkierungen 54

4 Schriftenkatalog

1	Schriftfamilien	55
2	Schriftanwahl	55
3	Vergrößerung — global	57
4	Fonts in eigenen Vergrößerungen	57
5	Vorhandene Schriften	58
6	“Computer Modern” Schriften	58

5 Mathematischer Formelsatz

1	Vorbemerkung	65
2	Grundsätzliches	65
3	Griechische Buchstaben	66
4	Exponenten und Indizes	68
5	Wurzelzeichen	69
6	Kennzeichnung durch Akzente und Striche	69
7	Hervorgehobene Formeln — ‘display-style’	70
8	Brüche	71
9	Binomial-Koeffizienten	72
10	Integralzeichen, Summen und andere Operatoren	72
11	Klammern und Begrenzer	74
12	Wachsende Klammern	77
13	Mathematische Funktionen	78
14	Abstände	78
15	Matrizen	79
16	Einzelne gruppierende Klammern	82
17	Mehrzeilige Formeln — Ausrichtung	83
18	Numerierte Formeln	84
19	Mathematische binäre Operatoren	86
20	Mathematische Relationen	86
21	Sonstige mathematische Symbole	87
22	Leeroperatoren	88
23	Lange Formeln — Formeltrennung	89

6 Tabellensatz

1	Tabulatoren	91
2	Variable Tabulatoren	95
3	Automatischer Tabellensatz: die Musterzeile	95
4	Automatischer Tabellensatz: Spaltenausrichtung	98
5	Automatischer Tabellensatz: Spaltenabstand	99
6	Automatischer Tabellensatz: gerahmte Tabellen	102
7	Hilfsmittel beim Tabellensatz	106

7 Eigene Definitionen und Befehle — Makros

1 Einfache Makros 109
2 Makros mit Parametern 110
3 Makros innerhalb von Makros 112
4 Expandierung von Makrobefehlen 112
5 Abfragen 114
6 Eigene if-Befehle 117
7 Testen der Makros 118
8 Trick-Makros oder Makros für Fortgeschrittene 119

8 Wie TEX arbeitet

1 Kästchen 129
2 TEX's interne Arbeitsmodi 129
3 Box-Manöver 130
4 Box-Register 132
5 Box-Ausgaben 132
6 Box-Dimensionen 134
7 Einrahmungen 136
8 Blöcke und Schatten 139
9 Dynamischer Leerraum 142
10 Box-Bewegungen 143
11 Box-Teilausgaben 145
12 Zähl- und Längenregister 147
13 Token Register 148

9 Variationen des Formelsatzes

1 Abstandssteuerung 151
2 Eigene mathematische Symbole 154
3 Layout-Veränderungen 156
4 Simulation von Exponenten und Indizes 158
5 AMS-Fonts Version 2.0 158

10 Fehlermeldungen

1 Format der Fehlermeldungen 163
2 Verhaltensweisen bei aufgetretenen Fehlern 164
3 Häufige Fehler und ihre Ursachen 164
4 Protokollparameter 170

11 Output-Routinen

1 Aufgabe einer Output-Routine 171
2 Die Standard-Output-Routine 172
3 Variationen der Output-Routine 175
4 Seitenspezifische Textmarkierungen 177

12 Anwendungsbeispiele

1 Balkendiagramme 179
2 Deutsche Anführungszeichen 183
3 Vorlesungsskript 185

13 Datenorganisation

1 Standarddateien 191
2 Organisation der Eingabe 192
3 INITEX 193
4 Zugriff auf weitere Klartextdateien 194
5 Eingaben aus Klartextdateien 195
6 Ausgaben in Klartextdateien 196

14 Anhang

1 Kurzbeschreibung der plain-TEX-Befehle 202
2 Versteckte plain-TEX-Befehle 307
3 Schlagwortregister 316
4 Fonttabellen der Standardschriften 322
5 Erweiterte 256-Zeichen TEX-Codebelegung 331
6 Literaturverzeichnis 335

1 Allgemeines

1.1 Charakteristika des TEX-Systems

TEX (sprich: "Tech", kann auch "TeX" oder "$\tau\epsilon\chi$" geschrieben werden) ist ein Satzsystem, das Donald E. Knuth an der Stanford University im Laufe eines Jahrzehnts entwickelte. Es ist insbesondere für die Erstellung wissenschaftlicher Veröffentlichungen, die mathematische Formeln enthalten, geeignet.

TEX ist ein *Satz*-System und *kein* Textsystem. Die Betonung liegt hier wirklich auf "Text setzen". Dieses Programm bietet so ziemlich den 'Stand der Kunst' im Bereich des computerunterstützten Textsatzes. Es hat eine Reihe von Fähigkeiten, die kein normales *Text*system besitzt. Kurz gesagt, bietet TEX folgende besonderen Vorzüge:

verschiedene Schriften TEX verwendet verschiedene Schriftarten und Schriftgrößen, im Normalfall in Proportionalschrift. Diese Schriften gehören zu der neu entwickelten Schriftfamilie "computer modern fonts". Insbesondere werden im Mathematiksatz die Schriftgrößen für Indizes, Subformeln usw. automatisch bestimmt. Inzwischen sind eine Reihe zusätzlicher Schriften verfügbar geworden.

Unterschneiden Es wird ein Unterschneiden *(kerning)* durchgeführt: Zeichen, die sich überlappen, werden etwas näher aneinandergerückt: So wird z.B. "Vor" bzw. "VA" und nicht "Vor" bzw. "VA" gesetzt.

Ligaturen Für eine Reihe von Zeichenfolgen — fi, fl, ffi, ffl — sind *Ligaturen* vorhanden. Es wird ein besonderes Zeichen für die zusammengesetzten Zeichen — fi, fl, ffi, ffl — gesetzt. Dies geschieht automatisch.

Randausgleich Der Text wird exakt am rechten Rand ausgeglichen. Die Zeichen stehen 'punktgenau' untereinander. Beim Textumbruch wird nicht nur der Inhalt einer Textzeile, sondern das Aussehen des gesamten Absatzes berücksichtigt. Es wird zum einen versucht, so wenig wie möglich zu trennen, zum anderen aber den Ausgleich zwischen den einzelnen Wörtern gleichmäßig zu gestalten.

Trennen Getrennt wird mit Hilfe von gespeicherten Trennmustern. Dieses Verfahren hat sich in der praktischen Anwendung bewährt.

Sein Schwergewicht liegt in der Vermeidung von falschen Trennungen, was sich mit dem Bemühen, Trennungen überhaupt zu vermeiden, gut kombiniert. Standardmäßig ist eine amerikanische Trenntabelle geladen. Deutsche Trennmuster sind ebenfalls erhältlich. Seit Version 3 kann innerhalb eines Dokumentes die Sprache, das heißt die Trennregeln, nach denen getrennt wird, gewechselt werden.

Mathematiksatz Mathematische Formeln werden in einer komfortablen Weise eingegeben und in einer Ausgabeform dargestellt, wie sie Mathematiker in gutem Buchsatz gewohnt sind. Die Eingabeform spiegelt die mathematische Struktur und Bedeutung der einzelnen Formelelemente wider.

Makros TEX beinhaltet einen komfortablen Makroprozessor — fast eine eigene Programmiersprache. Dies bietet dem Benutzer die Möglichkeit, eigene Befehle zu definieren, TEX-Befehle umzubenennen, mit neuen Eigenschaften zu versehen, ja seine ganz private Befehlswelt zu schaffen.

Layout Ganz allgemein gesprochen wird besonderer Wert auf die Gestaltung des Dokuments gelegt. Intelligente Algorithmen unterstützen Umbruch und Layout. Dies gestattet es dem Benutzer, mit TEX Dokumente in Druckqualität zu erzeugen. Dieses entstandene Layout eines Dokuments ist voll parametrisiert und kann durch den Bediener gesteuert werden.

Eingabe Die Eingabe ist für den Benutzer — solange er keine Formeln setzt — aufwendiger als bei normalen einfachen Textsystemen. Der Benutzer sollte sich im klaren sein, daß er mehr Aufwand treibt, dafür aber auch ein **viel** schöneres Ergebnis bekommt.

Ausgabe Die Ausgabe der Ergebnisse ist voll befriedigend nur an hochauflösenden Laserdruckern oder ähnlichen Geräten möglich. 24-Nadeldruckern erzielen durchaus brauchbare Ergebnisse.

Implementierungen TEX ist auf nahezu allen Rechnern und Betriebssystemen implementiert. Selbst auf Personal Computern sind inzwischen Implementierungen erhältlich. Mit dem starken Leistungszuwachs dieser Rechnerklasse lassen sich auch große Dokumente problemlos mit kurzen Laufzeiten bearbeiten.

Das TEX-System ist im Prinzip “public domain software”. Sie steht jedem Anwender kostenlos und gebührenfrei zur Verfügung. Die einzigen Kosten, die auftreten, sind die Erstellungskosten der Datenträger für die Quellübernahme. Allerdings ist es so, daß ja ein Programm — selbst wenn es in einer genormten Standardprogrammiersprache geschrieben ist — nicht ohne weiteres auf jedem Rechner und unter jedem Betriebssystem einfach läuft. Daher gibt es auch eine Reihe kommerzieller Implementierungen, die dem Anwender diese Arbeit ersparen.

2 Bedienung

2.1 Befehlsstruktur und Befehlszeichen

␣	0	@	P	‘	p	0
!	1	A	Q	a	q	1
"	2	B	R	b	r	2
#	3	C	S	c	s	3
$	4	D	T	d	t	4
%	5	E	U	e	u	5
&	6	F	V	f	v	6
’	7	G	W	g	w	7
(	8	H	X	h	x	8
)	9	I	Y	i	y	9
*	:	J	Z	j	z	A
+	;	K	[	k	{	B
,	<	L	\	l	|	C
-	=	M	]	m	}	D
.	>	N	^	n	~	E
/	?	O	_	o		F
2	3	4	5	6	7	

ASCII-Code (ohne Steuerzeichen)

TEX-Befehle und der normale Eingabetext werden aus *einer* gemeinsamen Eingabedatei gelesen und verarbeitet. Es gibt also *keine* Satzbefehle zu einem *externen* Text. Die möglichen Eingabezeichen, die — typischerweise in einer *Eingabedatei* — verwendet werden, orientieren sich an dem internationalen ASCII-Zeichensatz. Es wird also keine besondere Rechner- oder Terminaltastatur mit Sonderbelegungen vorausgesetzt, sondern eine normale Rechnertastatur mit internationaler Belegung. Dieser Zeichencode besitzt im Vergleich zu einer deutschen Schreibmaschine bzw. einer Tastatur mit deutscher Belegung an Stelle der Umlaute eckige und geschweifte Klammern. Die weiteren deutschen Symbole sind ‘ß’, dies ist durch eine Tilde ‘ ~ ’ ersetzt, und der Paragraph §, an dessen Stelle das ‘et-Symbol’ bzw. der ‘Klammeraffe’ @ steht. Dies führt bei der Eingabe deutscher Sonderzeichen zu Unhandlichkeiten.
Für TEX-Befehle und Befehlszeichen werden aus dem vorhandenen Zeichenvorrat solche Zeichen für Sonderfunktionen benutzt, die im normalen Text relativ selten auftreten.

Zunächst zu den TEX-Befehlen selbst:

\ Die **TEX-Befehle** beginnen mit einem Einleitungszeichen für den Befehlsnamen. Dies ist im Standard ein ‘\’ (backslash=inverser Schrägstrich). Auf dieses “escape”-Symbol folgt der Befehlsname. Dieser besteht im Normalfall nur aus Buchstaben,

es kann jedoch auch *ein* Nichtbuchstabe *(Kurzbefehl)* für den Befehlsnamen verwendet werden. Dazu gehören auch die Ziffern.
Die Befehle, die aus Buchstaben bestehen, werden von einem Blank oder dem nächsten Sonderzeichen abgeschlossen. Soll unmittelbar nach einem Namensbefehl ein Leerzeichen *ausgegeben* werden, muß nach dem Befehl die Folge \␣ stehen. (␣ steht dabei für ein Leerzeichen).

Beispiele für Namensbefehle	*Beispiele für Kurzbefehle*
`\smallskip`	`\"`
`\footnote`	`\,`
`\par`	`\_`
`\indent`	`\%`
`\ss`	`\-`

Beispiel für eine TEX-Eingabe:

```
{\bf \narrower \narrower \noindent
 Die Buchdruckerei ist eine so edle Kunst, da\ss\ man bei denen,
 die sie aus\"uben, einen gewissen Grad von Kultur voraussetzen
 sollte.\par \hfill {\it Johann Friedrich Unger\par}}
```

Ausgabe:

> **Die Buchdruckerei ist eine so edle Kunst, daß man bei denen, die sie ausüben, einen gewissen Grad von Kultur voraussetzen sollte.**
>
> *Johann Friedrich Unger*

Neben den *Befehlen* gibt es einige **Befehlszeichen**, die schon für sich bestimmte Wirkungen haben:

{ Blockklammern

} Befehle, insbesonders Einstellungen gewisser Parameter wie Schriftform, -größe, Einrückungen, sollen nur in bestimmten **Gültigkeitsbereichen** gelten.
Solch ein Bereich wird durch das Zeichen '`{`' eingeleitet und durch '`}`' beendet. Dies entspricht der in modernen Programmiersprachen üblichen Blockstruktur.
Alle Einstellungen, die innerhalb eines solchen Klammerpaares gemacht werden, sind außerhalb nicht mehr vorhanden. Es gelten dann die letzten Einstellungen, die außerhalb dieses Blocks gesetzt wurden.

& Das Tabulatorsymbol wird für die Erstellung von Tabellen benötigt. Es ist das 'TAB'-Zeichen und trennt die einzelnen Tabellenelemente.

% Kommentarsymbol: Alles, was einem Prozentzeichen in einer Zeile folgt, ist ein Kommentar und wird vom TEX-Programm ignoriert.

$ Mit Dollar beginnt und endet der mathematische Formelsatz. Hier nur ganz kurz einige Bemerkungen dazu, mehr Information finden Sie im Abschnitt *Formelsatz*.
Die Information zwischen einem Dollar- oder einem Doppel-Dollar-Paar wird als mathematische Formel betrachtet und nach eigenen Regeln gesetzt, also:

$ *Satzinformation für Formeln im Text* $
$$ *Satzinformation für hervorgehobene Formeln* $$

Parametersymbol
Dieses Zeichen ist ein *Ersetzungssymbol* bei der Definition von Makros (Abkürzungen) und beim Tabellensatz.

^ *Dach* für Superscript oder Exponenten in mathematischen Formeln
Zum Beispiel: `$x^2$` liefert x^2. Der Ersatzbefehl hierfür ist "`\sp`".

_ *Unterstrich* für Indizes in mathematischen Formeln
Zum Beispiel: `$x_n$` liefert x_n. Als Ersatzbefehl kann "`\sb`" verwendet werden.

- *Bindestrich*
Der einfache Bindestrich, der auf einer normalen Schreibmaschine zu finden ist, entspricht nicht den 4 Arten von 'Strichen', wie der Setzer sie kennt. Je nach Bedeutung werden ein bis drei Minuszeichen *eingegeben* für folgende Gruppen:

der *Bindestrich*	`O-Beine`	*für*	O-Beine
der *bis-Strich*	`12--14 Uhr`	*für*	12–14 Uhr
der *Gedankenstrich*	`--- nicht wahr ?`	*für*	— nicht wahr ?
das *mathematische Minuszeichen*	`$x-y$`	*für*	$x - y$

Will man diese Symbole selbst darstellen, ist eine Umschreibung notwendig:

`$\backslash$` `$\{$` `$\}$` `\$` `\&` `\#` `\_` `\%`

Die Zeichen ˆ und ˜ sind als *Akzente* verfügbar.

Die Zeichen `| < >` sind in der Schriftart "`typewriter type`" verfügbar. Allerdings treten kleiner-größer-Zeichen praktisch nur in Formeln auf. Im Normaltext kann man sie auch durch kurzen Übergang in den Mathematiksatz erzeugen: `$|$`, `$<$` und `$>$` erzeugen die Zeichen |, < und >.

2.2 Anwendung der Befehle

Bei den Befehlen lassen sich drei Gruppen unterscheiden:
• Befehle, die etwas einstellen
• Befehle, die auf etwas wirken
• Befehle, die für sich eine Wirkung haben
Befehle, die etwas einstellen, zum Beispiel eine bestimmte Schrift, werden stets in Zusammenhang mit Blockklammern verwendet, die die Gültigkeit dieser Einstellungen festlegen. Diese Klammern stehen um die Befehle und den Gültigkeitsbereich herum. Diese Struktur entspricht dem Blockkonzept einer Programmiersprache.

Ein Beispiel dazu:

`\rm` stellt die Normalschrift ein,
`\it` stellt die *Italicschrift* ein,
`\bf` stellt die **Fettschrift** ein.

Dann erhält man aus der Eingabe

```
Dieser Text ist in Normalschrift, {hier auch noch, \bf aber
jetzt in fett {\it mit etwas italic} etwas fett} und zum
guten Ende wieder normal.
```

das Ergebnis:

> Dieser Text ist in Normalschrift, hier auch noch, **aber jetzt in fett**
> *mit etwas italic* **etwas fett** und zum guten Ende wieder normal.

Die zweite Gruppe der Befehle, die auf etwas wirken, die also Parameter besitzen, besteht aus Befehlen wie

`\matrix{...}`	zum Satz von Matrizen
`\line{...}`	zum Satz einer Zeile
`\halign{...}`	zum Tabellensatz

Bei ihnen folgt — meist in Blockklammern — die zu verarbeitende Information. Wenn die Parameterinformation nur aus *einem* Zeichen oder *einem* weiteren Befehl besteht, dürfen die Klammern auch entfallen.

Beispiel: `$x_i$` für x_i und `$x_{ij}$` für x_{ij}

Die letzte Gruppe sind die parameterlosen einfachen Befehle ohne Umschaltungseffekte:

`\quad`	für etwas Leer raum
`\%`	für das %-Zeichen

2.3 Satzzeichen

Satzzeichen werden durch TEX speziell behandelt. Insbesondere wird am Satzende nach dem Punkt ein *etwas* breiterer Platz gelassen. Folgt der Punkt einem Großbuchstaben, hält TEX dies für eine Abkürzung und macht normale Abstände. Will man nach einem Punkt einen normalen Abstand gesetzt haben, ist `.\␣` einzugeben. Im europäischen Raum war diese Form der Abstandsbildung nicht üblich. Durch den Befehl `\frenchspacing` kann global eingestellt werden, daß die Leerzeichen auch am Satzende die gleiche Größe wie im Satz haben. Dieser Befehl läßt sich durch `\nonfrenchspacing` wieder rückgängig machen.

Eine häufige Anwendung sind drei aufeinander folgende Punkte ... zur Darstellung von "und so weiter". Die normale Eingabe `...` erzeugt "...". Besser ist es, den Befehl `\dots` zu verwenden, die Punkte sehen dann so aus "…".

`` ` `` Das Akzentzeichen der Tastatur erzeugt ein *linkes* hochgestelltes Apostroph:
— ‘ — (Ersatzbefehl ist "`\lq`" — *left quote*)
2 Akzentzeichen werden als Ligatur zusammengerückt: ` `` ` erzeugt “.

`'` Das Apostroph der Tastatur erzeugt ein *rechtes* hochgestelltes Apostroph:
— ’ — (Ersatzbefehl ist "`\rq`" — *right quote*)
2 Apostrophe werden als Ligatur zusammengerückt: `''` erzeugt ”.

`"` Ein Doppelapostroph hat als Ausgabe — ” —.
Man erhält in den proportionalen Schriften das gleiche Bild wie bei 2 einzelnen Apostrophen als Eingabe.

Für spezielle deutsche "Gänsefüßchen" werden im Abschnitt 12.2 geeignete Befehle vorgestellt, mit denen deutsche Anführungszeichen konstruiert werden. Ab TEX Version 3 und mit Verwendung der erweiterten 256-Zeichen TEX-Codebelegung stehen in neuen speziellen Zeichensätzen eigene Zeichen für deutsche Anführungszeichen zur Verfügung.

2.4 Motivierendes Beispiel

Im folgenden Beispiel sind die TEX-Befehle `negativ` gedruckt. Einzelne TEX-Befehle sind im obigen Beispiel durch einen winzigen Leerraum ▮▮ getrennt. Dies entspricht *keinem* Leerzeichen in der Eingabe. Es werden folgende Befehle verwendet:

`\"`	deutscher Umlaut folgt
`\ `	Erzwingen eines Leerzeichens
`---`	Gedankenstrich setzen
`\bf`	Standard-Schrifttype **boldface** einschalten
`\bigskip`	großen vertikaler Abstand (Leerzeile) setzen
`\centerline`	Zentrieren einer Zeile
`\dag`	das Symbol †
`\Delta`	Mathematisches Symbol Δ
`\end`	Ende-Anweisung an TEX
`\font`	Definition einer neuen Schrift
`\footnote`	Fußnote bilden
`\gross`	Anwahl des eigenen neuen Schrifttyps
`\headline`	Definition der permanenten Seitenüberschrift
`\hfill`	dynamischer horizontaler Leerraum
`\hrule`	waagerechter Strich
`\it`	Standard-Schrifttype *italic* einschalten
`\leftline`	linksbündige Zeile setzen
`\magnification`	globale Vergrößerung
`\magstep1`	Vergrößerung 1.2
`\magstep2`	Vergrößerung $1.44 = 1.2 * 1.2$
`\medskip`	mittlerer vertikaler Abstand
`\mittel`	Anwahl des eigenen Schrifttyps
`\noindent`	Absatzanfang *ohne* Einrückung bilden
`\over`	mathematische Operation für den Bruchstrich
`\pageno`	Einstellung der aktuellen Seitennummer
`\rightline`	rechtsbündige Zeile setzen
`\sl`	Standard-Schrifttype *slanted* einschalten
`\smallskip`	kleiner vertikaler Abstand
`\to`	mathematisches Operationszeichen $\rightarrow$
`\vfill`	dynamischer vertikaler Leerraum (Auffüllen)
`\vskip`	vertikalen Leerraum mit angegebener Länge setzen
`$$`	Anfang/Ende Mathematik
`{ }`	Gruppenklammern

```
\pageno=128
\magnification=\magstep0
\font\mittel=cmbx10 scaled \magstep1
\font\gross=cmbx10 scaled \magstep2
\headline={\hfill --- Beispiel --- \hfill}
\centerline{\gross Haupt\"uberschrift}
\bigskip
\leftline{\mittel Kapitel\"uberschrift}
\medskip
\leftline{\bf Abschnitts\"uberschrift}
\medskip
\noindent Dieses Beispiel wurde in einer 10 Punkt
hohen Schrift gesetzt. Die Originalschrift wurde
nicht ver\"andert.
Die globale Vergr\"o\ss erung
besitzt den Faktor $ f=1$.
Lediglich Zwischentitel sind in gro\ss en Schriften
gesetzt.
\smallskip
Textteile, die ich hervorheben m\"ochte, schreibe
ich in einer anderen Schrift: {\it italic}.
Es steht noch eine weitere schr\"age Schrift
--- {\sl slanted}--- zur Verf\"ugung.
\bigskip
\leftline{\bf Noch ein neuer Abschnitt}
\medskip
\noindent Ein neuer Abschnitt entsteht durch
eine oder mehrere Leerzeilen.
\bigskip
\rightline{\bf Die Absatz\"uberschrift mal auf der
    anderen Seite}
\medskip
Formelschreiben\footnote{\dag}{\TeX\ ist ja
auch in erster Linie f\"ur den mathematischen
Satz konzipiert.}geht durch \TeX\ ganz leicht:
$${
f( x + \Delta x )-f(x)
\over \Delta x }\to f'(x) $$
\bigskip
\hrule height 2pt \vskip 3pt \hrule
\bigskip
\centerline{\gross ENDE}
\vfill
\end
```

Beispieleingabe

Hauptüberschrift

Kapitelüberschrift

Abschnittsüberschrift

Dieses Beispiel wurde in einer 10 Punkt hohen Schrift gesetzt. Die Originalschrift wurde nicht verändert. Die globale Vergrößerung besitzt den Faktor $f = 1$. Lediglich Zwischentitel sind in großen Schriften gesetzt.

Textteile, die ich hervorheben möchte, schreibe ich in einer anderen Schrift: *italic*. Es steht noch eine weitere schräge Schrift — *slanted* — zur Verfügung.

Noch ein neuer Abschnitt

Ein neuer Abschnitt entsteht durch eine oder mehrere Leerzeilen.

Die Absatzüberschrift mal auf der anderen Seite

Formelschreiben† geht durch TEX ganz leicht:

$$\frac{f(x+\Delta x)-f(x)}{\Delta x} \to f'(x)$$

ENDE

† TEX ist ja auch in erster Linie für den mathematischen Satz konzipiert.

2.5 Programmstart und Programmende

Auf fast allen Systemen wird das TEX-Programm durch ein `TEX`-Kommando gestartet. Wird das Programm interaktiv aufgerufen, so meldet es sich zunächst mit der Anfrage nach einem *format-file*, in dem seine Makros abgespeichert sind.

```
This is TeX, Version 3.1 (preloaded format=plain 90.12.31)
**
```

oder

```
This is TeX, Version 3.1 (no format preloaded)
**
```

Die Versionsangaben können natürlich verschieden sein.

Wird an dieser Stelle mit `\relax` geantwortet, so kommt das Standardmakropaket *plain-TEX* zur Verwendung. Genau betrachtet wird das Makropaket verwendet, welches schon als "preloaded" gemeldet wurde, beziehungsweise, falls kein Makropaket schon vorab geladen ist, wird automatisch *plain-TEX* geladen. Es können jedoch auch andere Makropakete angewählt werden. Immer dabei vorausgesetzt, jemand hat diese mit `INITEX` vorbereitet und zur Verfügung gestellt. Die Anwahl eines Makropakets geschieht über die Angabe des Dateinamens, dem ein "`&`" vorangestellt wird. So ist `&plain` gleichbedeutend mit dem Standardpaket.

Auf der anderen Seite kann bei der ersten Anfrage auch direkt der Dateiname mit den Eingabedaten angegeben werden. In diesem Fall wird dann auch das Standardmakropaket verwendet.

Auch die Kombination zuerst die Angabe des gewünschten Makropakets, dann in der gleichen Zeile die Bezeichnung der Eingabedatei ist möglich.

```
This is TeX, Version 3.1 (preloaded format=plain 90.12.31)
**\relax
*
```

Standardmakros — anschließend Dialog

```
This is TeX, Version 3.1 (preloaded format=plain 90.12.31)
**&latex
*
```

Makros aus 'LATEX'-format file — anschließend Dialog

```
This is TeX, Version 3.1 (preloaded format=plain 90.12.31)
**meinfile
*
```

Standardmakros — Eingabe aus Datei '`meinfile`'

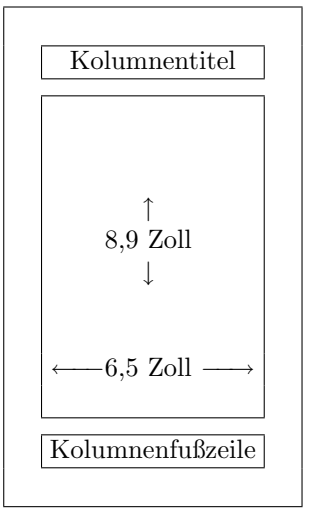
```
This is TeX, Version 3.1 (preloaded format=plain 90.12.31)
**&latex meinfile
*
```

Makros aus 'LATEX'-format file — Eingabe aus '`meinfile`'

Durch `\end` wird das Programm beendet, wobei ein "`\bye`" vorzuziehen ist. Dieses füllt nämlich die letzte Seite noch mit Leerplatz auf.

3 Textsatz

3.1 Das normale Druckbild



Das normale Seitenlayout, das voreingestellt benutzt wird, hat folgende Gestalt. Der Satzspiegel ist 8,9 Zoll hoch und 6,5 Zoll breit. Die Breite wird durch die interne Variable `\hsize` *(horizontal size)* und die Höhe durch `\vsize` *(vertical size)* repräsentiert. Eine Änderung dieser Werte, um einen anderen Satzspiegel zu erreichen, sollte tunlichst zu Beginn der Eingabe erfolgen. Gleichzeitig ist `\hsize` die Größe, nach der stets der Absatzumbruch erfolgt.

Über und unter den Text werden im Abstand einer Leerzeile (12pt) Kopf- und Fußzeile gesetzt. Die Kopfzeile ist laut Voreinstellung leer, die Fußzeile enthält zentriert die Seitennummer.

Ein normaler Fließtext ohne Besonderheiten wird seitenweise umbrochen und mit dem Standardlayout ausgegeben. Ein Text ist dabei normalerweise in Absätze unterteilt. In der Eingabe werden Absätze (Alinea) durch den Befehl `\par` *(paragraph)* oder durch eine leere Eingabezeile getrennt. In der Ausgabe hat ein solcher 'Normalabsatz' die Form, daß zu Beginn ein wenig Leerraum (Einzug) steht. Als günstig wird ein Einzug empfunden, der seitlich wie in der Höhe gleich ist, also ein optisches Quadrat bildet. Der Einzug im TEX-Programm ist etwas breiter: 20pt. Die zugehörige TEX-Variable heißt `\parindent`. Durch Zuweisungen wie beispielsweise `\parindent=15pt` kann der Einzug verringert bzw. durch `\parindent=30pt` vergrößert werden.

Gleichzeitig wird die Variable `\parindent` für den Anfangseinzug bei Aufzählungslisten (`\item`, siehe dazu 3.9) verwendet.

Soll ein Absatz — so wie dieser hier — beginnen, ohne daß am Anfang ein Einzug gesetzt wird, ist der Absatz mit dem Befehl `\noindent` zu beginnen. Das heißt, die

Eingabe für diesen aktuellen Absatz hat zu Anfang folgende Gestalt:

```
\par \noindent Soll ein Absatz --- so wie dieser hier ---
```

Die Alternative ist, `\parindent=0pt` zu setzen. Dies führt allerdings dazu, daß auch bei allen anderen Befehlen, die sich intern auf `\parindent` beziehen, `0pt` verwendet werden. Dies sind im wesentlichen `\item` und `\narrower`. Durch geeignetes Setzen der Gruppenklammern "`{`" und "`}`" lassen sich diese Seiteneffekte aber leicht verhindern.

Noch eine Warnung am Rande: Gibt man einen Absatz ein, der mehrere Seiten lang ist — an sich schon etwas ungewöhnlich — so kann die ein oder andere TEX-Implementierung damit Speicherprobleme bekommen: Das TEX-Programm liest nämlich den kompletten Absatz in seinen Datenspeicher, um den Umbruch zu optimieren.

3.2 Maßeinheiten

Längen können in TEX in verschiedenen Maßeinheiten angegeben werden. TEX selbst protokolliert seine Längenangaben in der Einheit "pt". Dies kommt von der Einheit der *Druckerpunkte*, im Englischen *point*. Die folgende Tabelle enthält die TEX bekannten verschiedenen Maßeinheiten.

Einheiten		**Umrechnungen**	
pt	*point* †		$1\,\text{pt} \approx 0{,}0351\,\text{cm}$
pc	*pica* †	$1\,\text{pc} = 12\,\text{pt}$	$1\,\text{pc} \approx 0{,}422\,\text{cm}$
in	*inch* †	$1\,\text{in} = 72{,}27\,\text{pt}$	$1\,\text{in} = 2{,}54\,\text{cm}$
bp	*big point* †	$72\,\text{bp} = 1\,\text{in}$	$1\,\text{bp} \approx 0{,}0353\,\text{cm}$
cm	*centimeter*		$1\,\text{cm} \approx 28{,}54\,\text{pt}$
mm	*millimeter*		$1\,\text{mm} = 0{,}1\,\text{cm}$
dd	*didôt point* ‡	$1157\,\text{dd} = 1238\,\text{pt}$	$1\,\text{dd} \approx 0{,}0376\,\text{cm}$
cc	*cicero* ‡	$1\,\text{cc} = 12\,\text{dd}$	$1\,\text{cc} \approx 0{,}451\,\text{cm}$
sp	*scaled point* ⋆	$65536\,\text{sp} = 1\,\text{pt}$	$1\,\text{sp} < 0{,}6 * 10^{-6}\,\text{cm}$
			† engl.-amerik. Pica-(Point)-System
			‡ deutsches Typographisches Punkt-System
			⋆ interne TEX-Einheit

TEX selbst rechnet mit ganzzahligen Vielfachen von "scaled point" (`sp`). Damit werden unterschiedliche Ergebnisse auf verschiedenen Rechnern durch Rundungsfehler vermieden. Der größtmögliche Länge, mit der TEX arbeiten kann, ist $1.073.741.823\,\text{sp} = (2^{30} - 1)\,\text{sp}$ oder umgerechnet etwa 5,7583 Meter. Dies sollte für normale Dokumente auch in Plakatgrößen ausreichen.

Neben den *festen* Maßeinheiten gibt es noch zwei Maßeinheiten aus dem Setzergewerbe, die von der aktuell eingestellten Schrift abhängen, also je nach Schriftwahl und Schriftgröße unterschiedliche Werte liefern:

em war früher als die Breite des großen 'M' definiert. Die Computer Modern Fonts haben die Eigenschaft, daß die Ziffern gerade 0.5 em breit sind.

ex ist in etwa die Höhe des kleinen "x"

Die Befehle `\quad` und `\qquad` erzeugen beispielsweise fontspezifischen Leerraum der Breite 1 em, bzw. 2 em.

Alle Einheiten, Abstände, Schriftgrößen sind noch von einem globalen Vergrößerungsfaktor, der `\magnification`, abhängig. Diese ist mit '1000' vorbesetzt, entsprechend dem Faktor '1,000'. Soll dieser Wert geändert werden, so darf dies nur zu Beginn der Eingabe geschehen, wenn noch keine einzige Länge aufgetreten ist. Eine zu späte oder doppelte Wertzuweisung wird mit einer Fehlermeldung abgelehnt. Prinzipiell kann jeder beliebige Wert auf `\magnification` zugewiesen werden, auch etwa `\magnification=0815` für den Faktor '0,815'. Das Problem dabei ist jedoch, ob hinterher bei der Ausgabe die entsprechenden Schriften für ihr Ausgabegerät vorhanden sind.

Ein Kompromiß in dieser Situation ist dabei die Beschränkung auf gewisse Abstufungen, und zwar jeweils Vergrößerungen um den Faktor '1,2'. Diese werden mit Hilfe der `\magstep`-Befehle eingegeben.

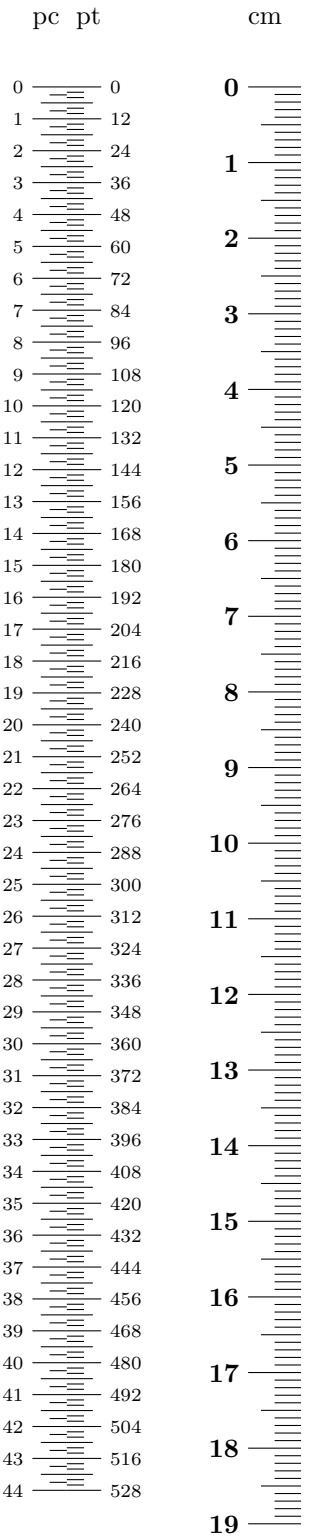
Angabe `\magnification=`	Faktor
`\magstep0`	1,000
`\magstephalf`	1,095
`\magstep1`	1,200
`\magstep2`	1,440
`\magstep3`	1,728
`\magstep4`	2,074
`\magstep5`	2,488

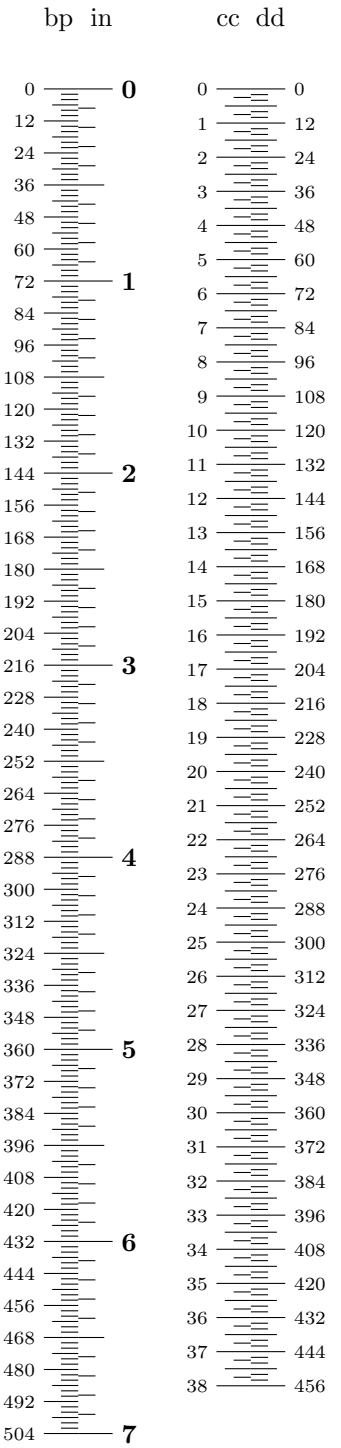
Die Aufrufe `\magstep0` ... `\magstep5` sind dabei nichts anderes als Abkürzungen für die Zahlenangabe '1000' ... '2488' (ohne Komma), wobei die Zahlen als Promilleangaben interpretiert werden.

Allerdings ist stets zu prüfen, ob bei der jeweiligen Implementierung die Fonts in den gewünschten Größen auch tatsächlich vorhanden sind. Als Minimum wird dabei im allgemeinen der Bereich `\magstep0` bis `\magstep3` empfunden.

Wird `\magnification` umgesetzt, so werden alle Größen, die nicht als 'true' deklariert sind, damit skaliert. Die Absätze werden mit größerem Zeilenabstand gesetzt. Es werden größere Symbole verwendet usw. Allerdings werden der Satzspiegel, Seitenhöhe (`\vsize`) und Zeilenlänge (`\hsize`) nicht verändert. Hier wird davon ausgegangen, daß das Ausgabegerät nicht ohne weiteres auch größeres Papier verarbeiten kann. Daher sind `\hsize` und `\vsize` belegt mit

```
\hsize = 6.5 true in
\vsize = 8.9 true in
```





Überhaupt kann man durch ein "`true`", das einer Längeneinheit vorangestellt wird, die Skalierung mit `\magnification` unterdrücken. Dies ist insbesondere für nachträglich einzumontierende Abbildungen mit festen Größen sehr sinnvoll.

3.3 Umlaute und nationale Sonderzeichen mit Version 2

Version 3: *Durch die Einführung von Version 3 gestaltet sich die Eingabe von Umlauten und Sonderzeichen einfacher — zumindest theoretisch, da einige Voraussetzungen erfüllt sein müssen. Jetzt ist es nämlich möglich, mit einem sogenannten 8-Bit-Eingabecode zu arbeiten und damit Umlaute direkt einzugeben. Allerdings steckt hier der Teufel im Detail. Voraussetzung ist nämlich, daß die vorliegende Implementierung auf dem jeweiligen Rechner das auch unterstützen muß. Nicht jeder Rechner ist in der Lage, 256 verschiedene Zeichen in der Eingabe zu akzeptieren. Zunächst soll daher die alte und immer funktionierene Form der Eingabe dargestellt werden:*

Deutsche Umlaute werden eingegeben, indem *vor* das Zeichen die Zeichenfolge `\"` geschrieben wird. Dies gilt für große wie auch für kleine Buchenstaben. Das deutsche "ß" wird durch die Zeichenfolge `\ss` dargestellt. Allerdings muß hier nach dem `\ss` ein Leerzeichen folgen, wenn das Wort anschließend weitergeht. Liegt das "ß" am Wortende vor, so sollte `\` und ein Leerzeichen* folgen. Also muß man für den Text "*Daß man großartig TEX kann*" folgendes eingeben

```
\it Da\ss\ man gro\ss artig \TeX\ kann
```

Ein Tip: Die Makrodefinition `\def\3{\ss}` bewirkt, daß man für 'ß' *immer* `\3` schreiben kann, ohne auf nachlaufende Leerzeichen und Wortenden zu achten. TEX erkennt an der Ziffer, daß der Befehlsname zu Ende ist. Das letzte Beispiel schreibt sich dann als:

```
\it Da\3 man gro\3artig \TeX\ kann
```

* Die Zahl der Leerzeichen zwischen zwei Wörtern ist übrigens egal. Mehrere werden wie ein Leerzeichen betrachtet.

Die Umlaute werden durch das TEX-Programm als normale Akzente oder diakritische Zeichen aufgefaßt. Die Symbole werden durch Komposition aus den Umlautpünktchen und dem nachfolgenden Zeichen konstruiert. Selbst folgende Zeichen sind möglich: Ä B̈ C̈! (Akzente im Mathematiksatz werden allerdings durch besondere Befehle erzeugt.) Es gibt 14 verschiedene Akzentbefehle in plain-TEX:

Eingabe	*Ausgabe*	
`\‘o`	ò	Gravis (Akzentzeichen)
`\’o`	ó	Akut (Apostroph)
`\^o`	ô	Zirkumflex
`\"o`	ö	Trema, deutscher Umlaut
`\~o`	õ	Tilde
`\=o`	ō	Querstrich, Macron
`\.o`	ȯ	Punkt-Akzent
`\u o`	ŏ	Halbkreis, breve accent
`\v o`	ǒ	Häkchen, háček
`\H o`	ő	Doppelakut, langer ungarischer Umlaut
`\t oo`	o͡o	tie-after accent
`\c o`	o̧	Cedille
`\d o`	ọ	dot-under accent
`\b o`	o̲	bar-under accent

Daneben gibt es einige nationale Sondersymbole, hier ist das deutsche ‘ß’ zu nennen, die durch Kombination anderer Zeichen nicht zu bilden sind.

`\oe`	œ	französisches œ
`\OE`	Œ	französisches Œ
`\ae`	æ	skandinavische æ-Ligatur
`\AE`	Æ	skandinavische Æ-Ligatur
`\aa`	å	skandinavisches a mit Kreis
`\AA`	Å	skandinavisches A mit Kreis
`\o`	ø	skandinavisches gestrichenes o
`\O`	Ø	skandinavisches gestrichenes O
`\l`	ł	polnisches ‘l’
`\L`	Ł	polnisches ‘L’
`\ss`	ß	deutsches ‘sz’
`\i`	ı	punktloses “i” (für Akzente)
`\j`	ȷ	punktloses “j” (für Akzente)

Die Akzentzeichen werden intern durch den `\accent` Befehl gesetzt. Dieser gibt den Code des Zeichens an, welches über das nachfolgende Symbol gesetzt werden soll. Es ist sogar möglich, das Akzentsymbol und das zu akzentuierende Zeichen aus verschiedenen Schriften* zu entnehmen. Beispielsweise wird durch `\bf\accent"7F\tenrm O` ein Umlaut Ö mit den Pünktchen aus der fetten Schrift gesetzt, zum Vergleich “ÖÖÖÖ”.

* Für Bastler ist das folgende Makro bestimmt:

```
\def\UMLAUT#1{{\edef\next{\bf\accent"7F\the\font#1}\next}}
```

Es werden stets die Umlautpünktchen aus der fetten Schrift verwendet, unabhängig davon welcher Font gerade eingestellt ist.

3.4 Umlaute und Akzente mit Version 3

Im folgenden sollen einige Rezepte angegeben, wie die Eingabe der Sonderzeichen vereinfacht werden kann. Dabei werden einige Befehle verwendet, deren Erläuterung hier noch nicht stattfindet.

Vorausgesetzt ihre TEX-Implementierung erlaubt, Umlaute direkt einzugeben, so sind zwei Fälle zu unterscheiden: Es werden die alten Computer Modern Fonts oder auf 256 Zeichen erweiterte Schriften verwendet. Werden die alten Schriften verwendet, so kann jedes Sonderzeichen über den Umweg eines Makros angesteuert werden:

Nachdem die Eingabesymbole als Makro ansteuerbar gemacht wurden

```
\catcode‘\ß=\active
\catcode‘\Ä=\active   \catcode‘\Ö=\active   \catcode‘\Ü=\active
\catcode‘\ä=\active   \catcode‘\ö=\active   \catcode‘\ü=\active
```

können die dazugehörigen Befehle definiert werden, die das Zeichen dann selbst konstruieren.

Für Computer Modern Fonts:

```
\letß=\ss
\defÄ{\"A}          \defÖ{\"O}          \defÜ{\"U}
\defä{\"a}          \defö{\"o}          \defü{\"u}
```

Beispielsweise für das “Erweiterte TEX Font Codierschema” (DC/EC Schriften), das im Abschnitt 14.5 näher beschrieben ist.

```
\chardefß="FF
\chardefÄ="C4       \chardefÖ="D6       \chardefÜ="DC
\chardefä="E4       \chardefö="F6       \chardefü="FC
```

wobei die hexadezimalen Angaben die Positionen in der Codetabelle angeben. Eine Belegungstabelle ist im Anhang zu finden.

Das hier geschilderte Verfahren ist natürlich auch für die anderen nationalen Zeichen anwendbar. Wenn beispielsweise mit einem PC gearbeitet wird, können so alle Zeichen, die direkt eingebbar und auch darstellbar sind, behandelt werden.

Am einfachsten haben es die Anwender deren Implementierung die Umkodierung einfach passend vom nationalen Sonderzeichen des Rechners auf das passende TEX Zeichen im 256-Zeichencode konvertiert. Insbesondere werden dann auch die Umlaute ausgewertet, die in den Trennpattern für die deutsche Sprache enthalten sind.

Eine Warnung!

Man beachte, daß TEX-Dateien, die solche Eingabedaten enthalten, *nicht portabel* sind und sich nur mit Schwierigkeiten zwischen verschiedenen Rechnersystemen austauschen lassen. Insbesondere lassen sich solche Dokumente praktisch überhaupt nicht mit elektronischer Post versenden. An dieser Stelle hat man leider die Wahl zwischen Komfort und Portabilität.

3.5 Abstände zwischen Absätzen

Zwischen den Absätzen wird normalerweise kein zusätzlicher Leerraum gelassen. Ist jedoch das dringende Bedürfnis vorhanden, so etwas zu erzeugen, kann mittels Umsetzung der Variablen `\parskip` dies automatisch erreicht werden. Standardmäßig ist diese besetzt mit

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

Der Ausdruck "`plus 1 pt`" bedeutet, daß dem TEX-Programm gestattet ist, beim Seitenumbruch bis zu 1 pt (≈ 0,3 mm) zusätzlichen Platz zwischen den einzelnen Absätzen zu lassen, um die Seite gleichmäßig aufzufüllen.

Soll ein expliziter Abstand jeweils automatisch eingefügt werden, so ist für einen Abstand von 3 pt (dies entspricht `\smallskip`) folgende Anweisung empfohlen:

```
\parskip=3pt plus 1pt
```

Eine solche Automatik ist an sich in der Anwendung nicht so interessant. Wichtig sind die folgenden, in der Praxis häufig wiederkehrenden Befehle:

`\smallskip`		
`\medskip`	= 2× `\smallskip`	
`\bigskip`	= 2× `\medskip`	= *1 Leerzeile*
`\vskip` *Längenangabe*		

Diese lassen jeweils folgenden Leerplatz:

für `\smallskip` — Höhe: `\magnification` ∗ 3 pt

für `\medskip` — Höhe: `\magnification` ∗ 6 pt

für `\bigskip` — Höhe: `\magnification` ∗ 12 pt = eine Leerzeile

für `\vskip 1.7 cm` — So kann man beliebig und exakt Platz lassen!

"pt" ist dabei die schon vorher erwähnte übliche Maßeinheit. Alle Abstände werden entsprechend der globalen Skalierung berechnet. Beim Befehl `\vskip` ist noch in irgendeiner Einheit der gewünschte Umfang des Platzes anzugeben.

Ein bisweilen nützlicher Befehl ist `\removelastskip`, der den vorangehenden Leerraum entfernt — natürlich nur, falls ein solcher vorhanden ist. In der Praxis wird dies in Makros für Überschriften etc. angewendet, bei denen man nicht weiß, ob Leerraum vor einer Überschrift extern schon gesetzt wurde.

Neben den bisher beschriebenen 'statischen' skip-Befehlen gibt es noch dynamische Varianten, die soviel Platz lassen 'wie es geht'. Dies sind `\vss`, `\vfil` und `\vfill`. Sie sind im Kapitel "Wie TEX arbeitet" (8.8) näher beschrieben.

Der Leerraum zwischen zwei Absätzen verschwindet, falls der Seitenumbruch gerade so fällt, daß der Anfang des zweiten Absatzes gerade auf den Beginn der neuen Seite fällt. Dadurch wird gewährleistet, daß der Seitenanfang immer schön gleichmäßig aussieht. Am Seitenanfang wird automatisch so viel Platz gelassen, wie die durch die Variable `\topskip` angegeben ist. Für Leerraum, der in *keinem* Fall verschwinden soll, existiert der Befehl "`\vglue` *dimension*", der die gleiche Syntax wie "`\vskip`" besitzt. Dieser zusätzliche Platz erscheint in jedem Fall auch am Anfang einer Seite. Ab Version 3 gibt es für diese Anwendung den besonderen Befehl `\topglue`, der den `\topskip` entfernt und exakt Platz läßt, wie durch die folgende Längenangabe gefordert ist.

Die bisher genannten Befehle `\smallskip`, `\medskip`, `\bigskip` und `\vskip` beenden den aktuellen Absatz, um den Leerraum zu setzen. Anschließend beginnt eventuell ein neuer Absatz. Etwas anderes ist es, *innerhalb* eines Absatzes Platz zu lassen, so daß der Zeilenumbruch um die Lücke herumläuft. Im aktuellen Absatz ist gerade dies

Hier wurde Platz gelassen.

geschehen. Der Befehl `\vadjust` bietet diese Leistung. Steht der Umbruch eines Absatzes fest, so wird die als Parameter angegebene Information *nach* der Zeile eingefügt, in der der `\vadjust`-Befehl steht. Die Eingabe für dieses Beispiel lautet:

```
 ... die L\"ucke heruml\"auft.
\vadjust{\vskip 0.5cm
         \centerline{Hier wurde Platz gelassen.}
         \vskip 0.5cm}
Im aktuellen Absatz ...
```

So kann etwa auch durch ein einfaches "`\vadjust{\smallskip}`" einmalig ein erweiterter Zeilenabstand gesetzt werden.

3.6 Abstände zwischen Wörtern

Horizontale Abstände oder Leerräume werden meist in Vielfachen der Einheit "`em`" gesetzt. Diese entspricht der Breite des großen 'M', sie wird auch als 'Druckerviertelchen' bezeichnet. Die üblichen Befehle, um in einem laufenden Text Leerraum einzusetzen, lauten dann:

Befehl	*Abstand*	*Einheit*	*Bemerkung*
`\indent`		20 pt ≡ `\parindent`	Absatzeinrückung
schriftabhängige Abstände			
`\␣`			Leerzeichen
`\enskip`		0.5 em	
`\quad`		1 em	'Druckerviertelchen'
`\qquad`		2 em	

Gelegentlich soll Leerraum gesetzt werden, der nicht umbrochen werden soll. Durch "`~`" (Tilde) kann ein "geschütztes Leerzeichen" eingegeben werden, so etwa in der Angabe

"`A.~B.~Genius`". Der Name wird damit am Zeilenende nicht mehr an den Leerstellen aufgebrochen.

Neben dem Befehl "`\hskip` *dimension*", der analog zu "`\vskip`" versorgt wird und beliebig Leerraum zu setzen erlaubt, sind noch einige weitere Befehle vorhanden, die Leerraum setzen, aber keinen Umbruch zulassen.

Befehl	*Abstand*	*Einheit*	*Bemerkung*
`~`	| |		geschütztes Leerzeichen (Tilde)
`\enspace`	| |	0.5 em	Breite wie `\enskip`
`\thinspace`	|	1/6 em	

Zu `\thinspace` gibt es negatives Gegenstück "`\negthinspace`", mit dem wieder etwas Leerraum entfernt werden kann.

Eine weitere Anwendung für den Satz von Leerraum ist die Ausgabe in einer Länge, wie sie einem vorgegebenen Textstück entspricht. Diese Leistung erbringt der Befehl "`\phantom{ .. text .. }`". *Beispiel:*

```
\leftline{Jan Tschichold: Ausgew\"ahlte Aufs\"atze ...}
\leftline{\phantom{Jan Tschichold:} Birkh\"auser Verlag, Basel 1975}
```

liefert

Jan Tschichold: Ausgewählte Aufsätze ...
Birkhäuser Verlag, Basel 1975

Häufig werden diese Befehle mit den Anweisungen für eine zeilenweise Ausgabe `\leftline`, `\rightline` und `\centerline` kombiniert. Diese geben jeweils eine Zeile linksbündig, rechtsbündig oder zentriert aus. Sie können aber nicht innerhalb eines Absatzes verwendet werden.

Die bisher erwähnten Befehle geben Möglichkeiten, expliziten Leerraum zu setzen. Wie wird nun der durch den Randausgleich gebundene Wortzwischenraum geregelt?

Jede Schrift besitzt unter anderem vier Parameter, die angeben, wie breit ein Leerzeichen normalerweise ist, um welchen Betrag es sich ausdehnen und um wieviel es kleiner werden darf. Diese geben dem Programm die zum Umbruch notwendigen verschieden breiten Leerstellen. Es sind die Fontparameter

Parameter	*Bedeutung*	*Belegung in pt bei*			
		`\rm` `\sl`	`\bf`	`\ti`	`\tt`
`\fontdimen2`	normale Leerzeichenbreite	3.33	3.83	3.58	5.25
`\fontdimen3`	möglicher Zuwachs	1.67	1.92	1.53	0.00
`\fontdimen4`	möglicher Schrumpfanteil	1.11	1.28	1.02	0.00
`\fontdimen7`	Zusatzplatz am Satzende	1.11	1.28	1.02	5.25

Diese können auch lokal überschrieben werden, wenn etwa ein etwas breiterer Ausgleich erlaubt sein soll. Ist die Variable `\spaceskip` von Null verschieden, so wird nämlich deren Wert für die Leerraumbildung verwendet. Beispielsweise werden durch

```
\spaceskip=3.33pt plus 3.34pt minus 1.11pt
```

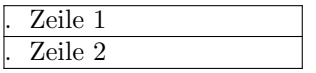
die Werte der Normalschrift mit doppeltem Zuwachsanteil eingestellt. Die Voreinstellung für den erweiterten Leerraum am Satzende wird mit `\xspaceskip` überschrieben. Beispielsweise vergrößert

```
\xspaceskip=3.33pt plus 2.22pt
```

den zusätzlichen Leerraum auf den doppelten Betrag.

3.7 Zeilenabstand

Der Zeilenabstand (Durchschuß) ist mit '12 pt' vordefiniert. Genau betrachtet ist dies die Entfernung, die die Grundlinien zweier aufeinanderfolgenden Zeilen haben sollen.



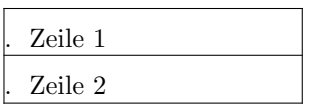
Der Abstand der Grundlinien — markiert durch "." — entspricht dem `\baselineskip`. Will man einen größeren oder kleineren Zeilenabstand haben, so kann man mit

```
\baselineskip=14pt
\baselineskip=10pt
```

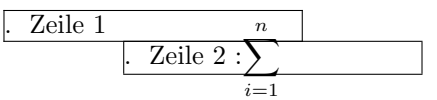
diesen vergrößern oder verkleinern. Eine andere Möglichkeit ist, durch

```
\baselineskip=1.5\baselineskip
\advance\baselineskip by 6pt
```

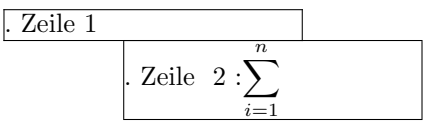
den aktuell eingestellten Wert zu verändern. Im folgenden Beispiel sind die Zeilenabstände um 6 pt vergrößert:



Die Veränderung des Zeilenabstandes betrifft übrigens auch mehrzeilige mathematische Formeln, die dann mit verändertem Abstand gesetzt werden.
Es kann nun sein, daß in der zweiten Zeile einige große Symbole vertreten sind:



Dies würde zu der so dargestellten Überlappung führen. Dagegen erzeugt das TEX-Programm in Wahrheit folgende Ausgabe:



In der ersten Beispielausgabe ragt die untere Zeile in die obere hinein. Um diesen Effekt zu verhindern gilt nun folgende Regel:

Ist der Abstand zwischen 2 Zeilen (Boxen) geringer als `\lineskiplimit`, dies ist der minimale Zeilenabstand, so wird als "Mindestabstand" ein Abstand von `\lineskip` zwischen der Unterkante der oberen Zeile und der Oberkante der unteren Zeile benutzt.

Dabei wird der Abstand zwischen den beiden Zeilen als Abstand zwischen den Grundlinien bestimmt.

Als Standard sind folgende Werte eingestellt:

```
\baselineskip=12pt
\lineskiplimit=0pt
\lineskip=1pt
```

Dies bedeutet praktisch, daß Überlappungen verhindert werden und stets mit einem Mindestabstand von 1pt gesetzt wird.
Übrigens, ein Überlappungseffekt kann auch durch zu große Zeichen in der vorangehenden Zeile verursacht werden, falls diese Zeichen mit sehr großen Unterlängen besitzt. Zeichen mit Unterlängen sind etwa 'y' oder 'g'. Dies führt gelegentlich bei Titelzeilen, in denen mit `\magstep` vergrößerte Schriften verwendet werden, zu genau diesem Effekt.

Zum Schluß noch einige Bemerkungen zur Wahl des Zeilenabstandes: Schriften mit kleineren Schriftgraden benötigen einen geringeren Zeilenabstand, fette und breite Schriften einen etwas höheren Zeilenabstand als normal. Folgende Werte sind bei den Standardschriften angemessen:

Schrifthöhe		`\baselineskip`
10 pt	z.B. cmr10	12 pt
9 pt	z.B. cmr9	11 pt
8 pt	z.B. cmr8	9 pt

Dies soll an der Normalschrift 'roman' und der fetten Schrift 'bold' demonstriert werden: Durch

```
\font\ninerm=cmr9
\font\eightrm=cmr8
\font\ninebf=cmbx9
\font\eightbf=cmbx8
```

werden die kleineren Schriftgrade definiert.

`\ninerm` mit `\baselineskip=11pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

`\eightrm`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

mit `\baselineskip=9pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

`\ninebf`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

mit `\baselineskip=11pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

`\eightbf`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

mit `\baselineskip=9pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, einen Aufbau gleich der Tektonik des Bauwerks.
Gustav Bartel

3.8 Zeilenausrichtung

Randausgleich und Flatterrand

Normalerweise werden alle Absätze rechts randausgeglichen, indem zwischen die einzelnen Wörter zusätzlicher Leerraum eingefügt wird. Soll der Text mit 'Flatterrand', also ohne Randausgleich gesetzt werden, so ist der Befehl `\raggedright` zu geben. Dadurch können die Leerräume zwischen den Wörten nicht mehr wachsen oder schrumpfen. Getrennt werden die Wörter am Zeilenende jedoch nachwievor. Mit den Gruppenklammern '`{`' und '`}`' kann die Gültigkeit von `\raggedright` auf bestimmte Bereiche eingeschränkt werden. Dabei ist jedoch zu beachten, daß das `\raggedright` noch gültig ist, wenn der Absatz zu Ende geht. Es ist also

```
{\raggedright ... text ... \par}
```

einzugeben und nicht etwa am Ende "`}\par`". Der aktuelle Absatz ist übrigens mit `\raggedright` gesetzt!

Soll ein Text, der in der Schrift '`\tt`' *(typewriter type)* ausgegeben wird, mit im "`\raggedright`" Modus gesetzt werden, so empfiehlt sich der Befehl `\ttraggedright`, da sonst die Leerzeichen nicht genauso breit wie die Textzeichen sind.

Es ist auch möglich, den Text links flattern zu lassen. Durch

```
\def\raggedleft{\leftskip=0pt plus 2em
          \spaceskip=0.333em
          \xspaceskip=.5em }
\def\ttraggedleft{\tt\leftskip=0pt plus 2em }
```

werden die dazu nötigen Befehle definiert. Dieser Absatz wurde mit `\raggedleft` behandelt. Damit sind alle Möglichkeiten, den Randausgleich links und rechts zu beeinflussen, besprochen. Im folgenden wenden wir uns nun Methoden zu, die Eingabe mehr oder weniger unverändert zu übernehmen. Dabei sollen zunächst die Zeilenstruktur und dann die komplette Eingabeform erhalten bleiben.

zeilenweise Ausgabe, Versmodus

In Gedichten werden die Texte natürlich zeilenweise ausgegeben. Damit man nun nicht mühevoll diese zum Beispiel mit eine Serie von `\leftline`-Befehlen eingeben muß, hilft der `\obeylines`-Befehl. Dieser bewirkt, daß die Zeilenenden jeweils als Absatzende interpretiert werden. Jede Zeile bildet dann für sich einen eigenen Absatz, beginnt also auch mit dem normalen Einzug (`\parindent`). Die Eingabe

```
{\obeylines
Gott segne
Kupfer, Druck und
jedes andere
vervielf\"altigende Mittel,
so da\3 das Gute,
was einmal da war,
nicht wieder
zu Grunde gehen kann.
\hfil Johann Wolfgang Goethe\par}
```

liefert

Gott segne
Kupfer, Druck und
jedes andere
vervielfältigende Mittel,
so daß das Gute,
was einmal da war,
nicht wieder
zu Grunde gehen kann.

Johann Wolfgang Goethe

Die `\obeylines`-Technik bewirkt also, daß jede Eingabezeile einen Absatz bildet. Genau betracht impliziert jedes Zeilenende einen `\par`-Befehl. Sind Eingabezeilen zu lang, so kann durch ein Kommentarzeichen '`%`' das Zeilenende 'wegkommentiert' werden. Dadurch wird die folgende Zeile die logische Fortsetzung der vorangehenden. `\obeylines` bleibt so lange gültig, bis der Block, in dem der `\obeylines`-Befehl steht, wieder zu Ende ist. Daher ist die typische Eingabeform für `\obeylines`:

```
{\obeylines
     ... text ...
}
```

Mit `\obeylines` können noch sehr bequem weitere Effekte erreicht werden:
`\everypar` speichert Befehle, die zu Beginn jedes Absatz automatisch ausgeführt werden. Fügt man in vorangehenden Beispiel ein "`\everypar{\hfil}`" hinzu

```
{\obeylines\everypar{\hfil}\parindent=0pt
Gott segne
   ...
```

so wird der Text zentriert gesetzt:

Gott segne
Kupfer, Druck und
jedes andere
vervielfältigende Mittel,
so daß das Gute,
was einmal da war,
nicht wieder
zu Grunde gehen kann.

Johann Wolfgang Goethe

Die Zentrierung kommt durch ein geschicktes Zusammenspiel mit der TEX-Variablen "`\parfillskip`" zustande. Diese regelt nämlich den Leerraum, der am Ende eines Absatzes automatisch gesetzt wird. Aufgrund der Belegung mit

```
\parfillskip=0pt plus 1fil
```

wird automatisch das Äquivalent von "`\hfil`" am Absatzende gesetzt. Da nun in der Folge von "`\everypar{\hfil}`" sowohl links als auch rechts jeden Absatzes ein "`\hfil`" steht, drückt dieser dynamische Leerraum das Ergebnis in die Mitte.

Selbst eine rechtsbündige Zeilenausrichtung funktioniert nach Hinzunahme der Befehle `\everypar{\hfill}`.

```
{\obeylines\everypar{\hfill}
Gott segne
Kupfer, Druck und
jedes andere
   ...
```

führt zu

Gott segne
Kupfer, Druck und
jedes andere
vervielfältigende Mittel,
so daß das Gute,
was einmal da war,
nicht wieder
zu Grunde gehen kann.
Johann Wolfgang Goethe

Die `\everypar`-Befehle definieren Befehlsfolgen, die zu Beginn jeden Absatzes *automatisch* eingefügt werden sollen. Da die Absätze hier jeweils nur aus einer Zeile bestehen, führt die Hinzunahme von `\hfil` oder `\hfill` zur Zentrierung oder zum rechtsbündigen Setzen.

'transparente Ausgabe'

Relativ häufig tritt das Bedürfnis auf, einen Eingabetext *völlig unverändert* auszugeben. Dies ist bei Programmquellen oder TEX-Eingaben zum Beispiel der Fall. Hier ist nun leider ein komplizierter Mechanismus nötig, der dies ermöglicht. Es müssen nämlich alle Steuerungen, die das TEX-Programm so hat, abgeschaltet werden, bis auf *einen*, der wieder alles ins Normale zurücksetzt. Die folgende Eingabe bewirkt genau dies:

```
\chardef\other=12 % d.h. sonstiges Zeichen
\def\ttverbatim{\begingroup
                \catcode`\\=\other \catcode`\{=\other
                \catcode`\}=\other \catcode`\$=\other
                \catcode`\&=\other \catcode`\#=\other
                \catcode`\%=\other \catcode`\~=\other
                \catcode`\_=\other \catcode`\^=\other
                \catcode`\|=\other
                \obeyspaces\obeylines\tt}
{\obeyspaces\gdef {\ }}
\outer\def\begintt{\let\par=\endgraf \ttverbatim \parskip=0pt
                  \ttfinish}
{\catcode`\|=0 |catcode`|\=\other
  |obeylines % Zeilenende wirkt wie \par
  |gdef|ttfinish#1^^M#2\endtt{#1|vbox{#2}|endgroup}}
```

Diese Befehle benutzen eine Reihe von Dingen, die bisher noch nicht erläutert wurden. Daher sei hier nur eine grobe Beschreibung der Funktionsweise dargestellt. Zunächst werden alle Steuerbefehle wie `$` und `\` abgeschaltet. Es bleiben hinterher zwei Befehle `\begintt` und `\endtt` übrig. `\begintt` schaltet in den transparenten Modus, und `\endtt` schaltet diesen wieder ab. Die Beispieleingaben in diesem Buch wurden übrigens auf genau diese Weise behandelt.

Die gesamte Information wird allerdings als Parameter in den Speicher eingelesen, so daß die Eingabe nicht zu lang sein darf. Auch muß die Ausgabe bei dieser Konstruktion noch zusammenhängend auf die Seite passen.

3.9 Absatzausrichtung

Der normale Textabsatz füllt die Zeile in ihrer gesamten Länge. Häufig werden jedoch Absätze mit zusätzlichem Leerraum auf der linken oder rechten Seite benötigt.

Vor und hinter jeder Zeile läßt das TEX-Programm automatisch extra Leerraum, nur daß dieser Platz mit 0pt initialisiert ist. Die beiden Kontrollvariablen heißen `\leftskip` und `\rightskip`. Wird wie für diesen Absatz `\leftskip=4cm` gesetzt, so wird vor jeder Zeile 4 cm zusätzlicher Platz gelassen. Damit verkürzen sich also automatisch die einzelnen Zeilen.

Auf der anderen Seite, wird Zusatzplatz mittels des Befehls `\rightskip=6cm` reserviert, bleibt halt auf der rechten Seitenhälfte etwas frei. Allerdings weiß man, daß die deutsche Sprache zu im Durchschnitt recht großen Wortlängen neigt und damit der Umbruch bei kurzen Zeilen nicht mehr sehr schön wird.
Auch hier gilt wieder die Bemerkung von oben, daß der Umbruch immer nach den am Absatzende gültigen Einstellungen erfolgt.

`\leftskip` und `\rightskip` lassen sich auch kombinieren. Neben der einzelnen Angabe erlaubt der zusätzliche Befehl `\narrower` eine inkrementale Erhöhung der aktuellen `\leftskip`- und `\rightskip`-Beträge. Durch `\narrower` wird jeweils der Betrag von `\leftskip` und `\rightskip` um den aktuellen Wert von `\parindent` erhöht.
`\parindent` ist die bereits erwähnte Größe des Einzuges zu Beginn jeden Absatzes.

Dieser Absatz wurde mit 1 × `\narrower` gesetzt und mit `\noindent` begonnen, um den Einzug zu unterdrücken. Das heißt, die Eingabe sieht im Prinzip so aus:

```
\par{\narrower\noindent Dieser Absatz wurde ... \par}
```

Der Befehl `\narrower` wirkt akkumulierend. Daher ist auf eine sorgfältige Benutzung der Blockklammern zu achten.

> Dieser Absatz wurde mit 3 × `\narrower` gesetzt. Damit erhält man also auf beiden Seiten den dreifachen Wert von `\parindent` als Rand.

Auch negative Angaben sind — mit Vorsicht ! — möglich. Der ganze Abschnitt verschiebt sich dann. Zum Beispiel verschiebt `\leftskip= 1 cm \rightskip= -1 cm`

den Abschnitt, ohne die Zeilenlänge zu verändern, um 1 cm nach rechts. Dies ist aber nur mit dem *ganzen* Absatz möglich, da der effektive Umbruch erst beim Absatzende festgestellt wird.
Man beachte allerdings, daß die Parameter `\leftskip` und `\rightskip` keinen Einfluß auf die Zentrierung einer hervorgehobenen mathematischen Formel, beispielsweise

$$\sum_{i=1}^{n} = \frac{n(n+1)}{2}$$

besitzen. Diese wird bezüglich der Originalzeilenlänge, also `\hsize` zentriert. Ein für eine Abbildung freizuhaltender Raum wird somit unter Umständen mitbedruckt.

\rightskip=0.5\hsize

Daneben gibt es noch Register, die die Gesamtverschiebung der Seite bei der Druckausgabe inkluse Kopf- und Fußzeilen regeln: Dazu zählt `\voffset`, das den Seitenanfang *bei der Ausgabe* vertikal verschiebt. Korrespondierend gehört dazu die TEX-Variable `\vsize`, die angibt, wie groß die Seitenlänge ist. `\vsize` ist die Größe, die TEX beim Seitenumbruch berücksichtigt. Kopf- und Fußzeilen werden dabei nicht berücksichtigt.

Will man beispielsweise seine Seite um 1 cm länger als normal erzeugen, so sind die folgenden Angaben sinnvoll:

```
\advance\vsize by 1 truecm
\voffset=-0.5 truecm
```

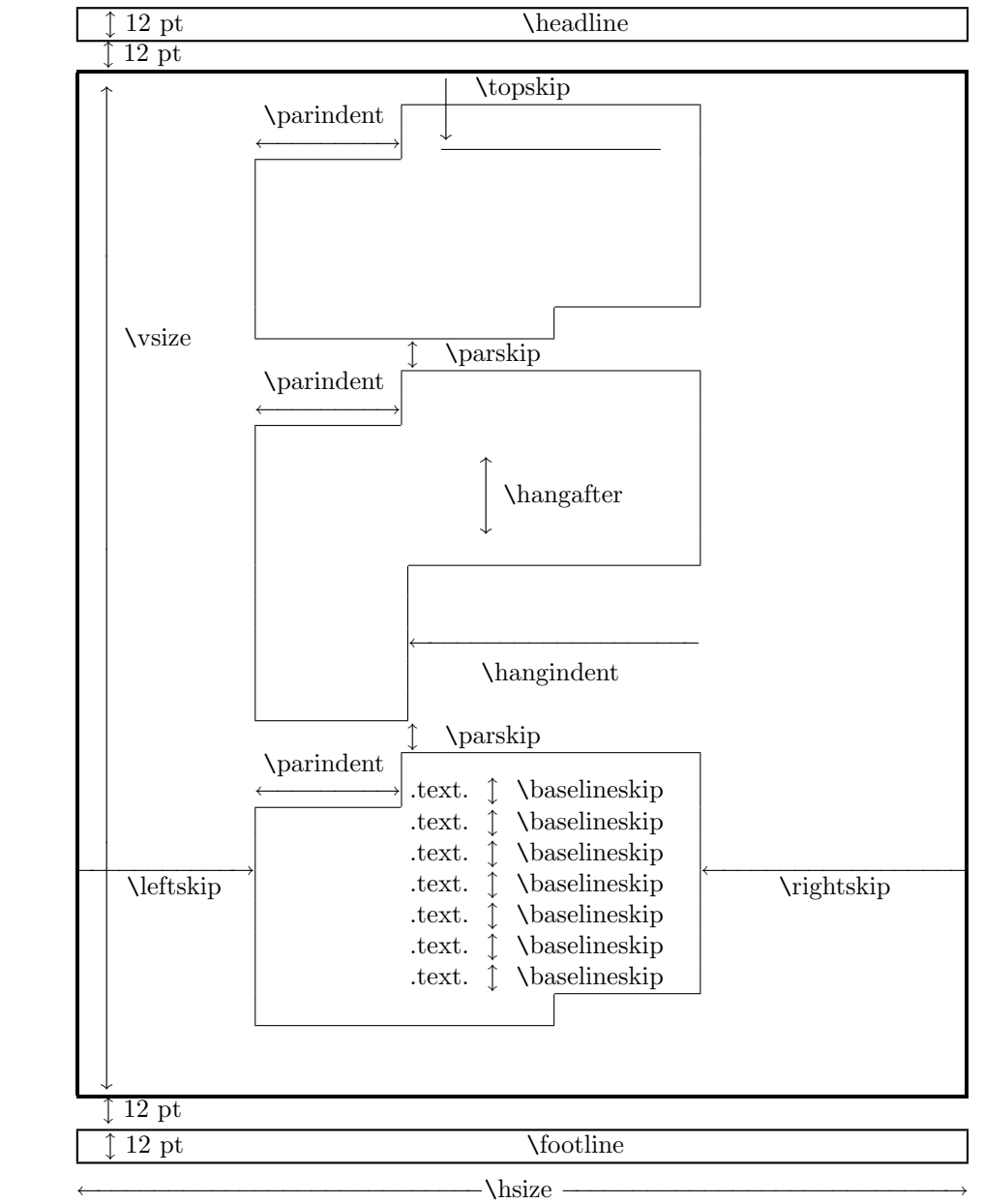
Dadurch wird die Seitenlänge um 1 cm vergrößert und die Seite immer noch vertikal zentriert ausgegeben. Genau betrachtet beginnt die Druckausgabe einen halben Zentimeter weiter oben und endet einen halben Zentimeter tiefer.

Die entsprechenden Größen für die horizontale Ausrichtung ist `\hsize` und für die horizontale Verschiebung bei der Ausgabe `\hoffset`.Das interne Register `\hsize` gibt die Textbreite einer Zeile an. Aufgrund von `\hsize` werden die Absätze in Zeilen umbrochen.
`\hoffset` gibt die horizontale Verschiebung des gesamten Ausdrucks bei der Druckausgabe an. Ein typischer Anwendungsfall ist die Verkleinerung der Seitenbreite und die Vergrößerung des linken Randes zum Ablochen oder ähnlichem.

```
\advance\hsize by -2 truecm
\advance\hoffset by 2truecm
```

Die Seiten werden um zwei *unskalierte* Zentimeter schmaler und nach rechts verschoben gedruckt.

Die globale Setzung von `\hoffset` und `\voffset` ist eigentlich in fast keinen Anwendungen nötig, da die Druckertreiber praktisch alle die Möglichkeit bieten, diese zu verändern. Einige sind sogar so komfortabel, daß für gerade und ungerade Seitenzahlen verschiedene Werte verwendet werden. Die Vorderseiten und Rückseiten können somit verschoben und dennoch deckungsgleich gedruckt werden.



Alle Parameter sind mit 0 bzw. '0 pt' vorbesetzt, bis auf:

\baselineskip = 12 pt	\hsize = 6.5 in	\topskip = 10 pt
\parskip = 0 pt plus 1 pt	\vsize = 8.9 in	
\parindent = 20 pt	\hangafter = 1	

Bedeutung der Positionierungsvariablen

3.10 Vielfache Absatzformen

Ein Absatz kann in der Praxis in vier verschiedenen Gestalten auftreten: Einrückungen links und rechts, oben und unten. Diese vier Formen lassen sich durch folgende zwei Befehle erreichen:

Die TEX-Variable `\hangindent` trägt immer den Wert der Einrückung. Ist der Wert größer als Null, wird links eingerückt, ist er kleiner als Null, wird rechts eingerückt. Der Wert kann in einer der vielen möglichen Maßeinheiten angegeben werden, z.B. `\hangindent=3cm` oder `\hangindent=0.4in`.

Die TEX-Variable `\hangafter` gibt an, wann die Einrückung anfängt bzw. aufhört. Ist `\hangafter` $= n$ und $n < 0$, so werden die ersten $|n|$ Zeilen eingerückt. Ist jedoch $n \geq 0$, so werden die ersten n Zeilen nicht eingerückt, sondern alle folgenden Zeilen.

`\hangindent=2cm \hangafter=-4`

Die **TEX-Befehle** beginnen mit einem \ (backslash), dann folgt der Befehlsname. Dieser besteht im Normalfall nur aus Buchstaben, es kann jedoch auch *ein* Sonderzeichen für den Befehlsnamen verwendet werden. (Ziffern sind in diesem Sinn auch Sonderzeichen.) Die Befehle, die aus Buchstaben bestehen, werden von einem Blank oder dem nächsten Sonderzeichen abgeschlossen. Soll unmittelbar nach einem Befehl ein Leerzeichen *ausgegeben werden*, muß nach dem Befehl die Folge \␣ stehen. (␣ steht dabei für ein Leerzeichen).

`\hangindent=2cm \hangafter=4`

Die **TEX-Befehle** beginnen mit einem \ (backslash), dann folgt der Befehlsname. Dieser besteht im Normalfall nur aus Buchstaben, es kann jedoch auch *ein* Sonderzeichen für den Befehlsnamen verwendet werden. (Ziffern sind in diesem Sinn auch Sonderzeichen.) Die Befehle, die aus Buchstaben bestehen, werden von einem Blank oder dem nächsten Sonderzeichen abgeschlossen. Soll unmittelbar nach einem Befehl ein Leerzeichen *ausgegeben werden*, muß nach dem Befehl die Folge \␣ stehen. (␣ steht dabei für ein Leerzeichen).

`\hangindent=-2cm \hangafter=-4`

Die **TEX-Befehle** beginnen mit einem \ (backslash), dann folgt der Befehlsname. Dieser besteht im Normalfall nur aus Buchstaben, es kann jedoch auch *ein* Sonderzeichen für den Befehlsnamen verwendet werden. (Ziffern sind in diesem Sinn auch Sonderzeichen.) Die Befehle, die aus Buchstaben bestehen, werden von einem Blank oder dem nächsten Sonderzeichen abgeschlossen. Soll unmittelbar nach einem Befehl ein Leerzeichen *ausgegeben werden*, muß nach dem Befehl die Folge \␣ stehen. (␣ steht dabei für ein Leerzeichen).

`\hangindent=-2cm \hangafter=4`

Die **TEX-Befehle** beginnen mit einem \ (backslash), dann folgt der Befehlsname. Dieser besteht im Normalfall nur aus Buchstaben, es kann jedoch auch *ein* Sonderzeichen für den Befehlsnamen verwendet werden. (Ziffern sind in diesem Sinn auch Sonderzeichen.) Die Befehle, die aus Buchstaben bestehen, werden von einem Blank oder dem nächsten Sonderzeichen abgeschlossen. Soll unmittelbar nach einem Befehl ein Leerzeichen *ausgegeben werden*, muß nach dem Befehl die Folge \␣ stehen. (␣ steht dabei für ein Leerzeichen).

Standardmäßig ist `\hangindent=0pt` und `\hangafter=1` gesetzt. Am Absatzende werden diese Werte automatisch wieder eingestellt.
Man beachte jedoch, daß nachwievor jeder Absatz mit einer Einrückung beginnt, wenn nicht — wie hier — `\noindent` am Anfang steht!

Die letzte
Möglichkeit, die Aus-
gabeform eines Absatzes zu
gestalten, ist Vorgabe der Zeilenlänge
für jede einzelne Zeile. Dabei muß man al-
lerdings schon sehr genau wissen, was eigentlich ge-
schehen soll. Meist funktioniert diese Technik nur mit ziem-
lich viel Probieren. Der Befehl `\parshape` definiert das Aussehen
eines Absatzes Zeile für Zeile. Seine Parameterversorgung geschieht mittels
der folgenden Syntax: `\parshape` $= n\ i_1\, l_1\ i_2\, l_2\ \ldots\ i_n\, l_n$. Dabei gibt der Parameter n an, für wieviele Zeilen Definitionspaare folgen. Jedes Definitionspaar besteht aus der Angabe "i_j" für den Einzug und der Längenangabe "l_j" für die entsprechende Zeile. Sind mehr als n Zeilen vorhanden, so wird die letzte Angabe stets weiter verwendet. Sind mehr Angaben als Zeilen vorhanden, so werden die überflüssigen Angaben ignoriert. Um das ganze zu illustrieren, ist dieser Absatz unter Anwendung von `\parshape` gesetzt worden, und zwar mit den folgenden Angaben für die ersten 10 Zeilen.

```
\parshape=10 0.45\hsize   0.1\hsize
             0.40\hsize   0.2\hsize
             0.35\hsize   0.3\hsize
             0.30\hsize   0.4\hsize
             0.25\hsize   0.5\hsize
             0.20\hsize   0.6\hsize
             0.15\hsize   0.7\hsize
             0.10\hsize   0.8\hsize
             0.05\hsize   0.9\hsize
             0.0 \hsize   1.0\hsize
```

3.11 Schmale Absätze und Umbruchsteuerung

Werden Absätze mit einer kurzen Zeilenlänge erzeugt, ist die Wahrscheinlichkeit sehr hoch, daß TEX eine "`Overfull \hbox`" meldet. Das ist eine Zeile, die zum Beispiel wegen fehlender Trennmöglichkeiten nicht wunschgemäß umbrochen werden kann. Schon vor TEX Version 3 konnte der Leerraum zwischen den Wörtern durch `\spaceskip` verändert werden. Beispielsweise `\spaceskip=3.33pt plus 4.44pt minus 1.11pt` läßt den Leerraum zwischen den Wörtern beim Umbruch weiter werden. Ab Version 3 ist es möglich, einfach durch Zuweisung auf die interne Variable `\emergencystretch` den Leerraum anzugeben, der innerhalb einer Zeile zusätzlich verteilt werden darf, wenn der normale Umbruch zu einer überlangen Zeile führt. In diesem Fall führt das TEX-Programm beim Absatzumbruch einen dritten Durchgang aus. Nur für diesen Durchgang wird `\emergencystretch`, falls der Wert größer als 0 pt ist, ausgewertet. Allerdings kann es hierbei zu Meldungen "`Underfull \hbox`" kommen, da die so entstandenen Zeilen trotzdem als unschön aufgefaßt werden. Der zusätzliche Leerraum wird nämlich als nicht existierend aufgefaßt, obwohl er gesetzt wird.

normal

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

`\spaceskip=`
`3.33pt plus 4.44pt minus 1.11pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

`\tolerance=10000`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

`\emergencystretch=`
`5pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

`\emergencystretch=`
`7pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

`\emergencystretch=`
`10pt`

Die Lettern, diese kleinen selbstverständlichen Zeichen, an denen Unzählige achtlos vorübergehen, gehören zu den bedeutendsten Formungen der menschlichen Schöpferkraft. Diese Gebilde, die wir alle täglich millionenmal mit unseren Augen aufnehmen, bannen die höchste Kunstfertigkeit in ihr kleines Format. ... Diese so lebendigen zugleich prägnanten Zeichen verlangen eine Gesetzmäßigkeit der Zueinanderordnung, ...

Damit ist es leicht möglich, den in Zeitungen häufig zu findenden sehr weiten Umbruch zu erzeugen. Da die "`Underfull \hbox`" Meldungen schon auf die Dauer lästig fallen können, kann man diese, zumindest lokal, mit `\hbadness=10000` vollständig abstellen. Durch beispielsweise `\hfuzz=3pt` werden zusätzlich nur noch überfüllte Boxen protokolliert, deren Überfüllung 3 pt übersteigt. Allerdings werden dann durch die `\overfullrule` auch nur noch diese Boxen markiert.

Wird auch noch der Parameter `\pretolerance=10000` gesetzt, so wird nicht mehr getrennt. Der gleiche Effekt wird allerdings auch durch Anwahl einer Sprache mittels `\language` erzielt, zu der keine Trennregeln geladen sind.

Alternativ zu `\emergencystretch` kann auch der allgemeine Parameter `\tolerance` hochgesetzt werden. Der Umbruch mittels `\emergencystretch` wird jedoch wesentlich besser als mit einer Einstellung von `\tolerance=10000` sein, da dann jeder noch so schlechter Zeilenfall akzeptiert wird. Mit großer `\tolerance` gesetzte Absätze können auch noch überlange Zeilen besitzen, mit einer genügend großen Angabe für `\emergencystretch` erzeugte Absätze jedoch nicht mehr.

3.12 Einrückungen und Listen

Einrückungen dieser Art geschehen mit dem `\item`-Befehl. Bis zum Ende des aktuellen Absatzes wird der Text um den Betrag von `\parindent` nach rechts gerückt. Es ist auch möglich, an den Anfang der Einrückung etwas zu schreiben. Wird also an den Anfang eines Absatzes der Befehl `\item{text}` gesetzt, so wird der Paragraph wie im folgenden eingerückt.

a) Hier wurde zum Beispiel am Paragraphanfang der Befehl `\item{a)}` gegeben. Dadurch erscheint die Zeichenfolge `a)` am Anfang.

   Durch den Befehl `\itemitem{ text }` wird eine doppelte Einrückung vollzogen.

   a_1 Wie man hier sieht. Es wurde ein Stück weiter eingerückt. Jedoch ist zu beachten, daß der Text, der am Anfang ausgerückt steht, nicht zu lang ist, da dieser *überlappend* geschrieben wird. Die Eingabe für diesen Absatz beginnt mit

   ```
   \itemitem{a$_1$}Wie man hier ...
   ```

   Hingewiesen sei noch auf die Feinheit, daß hier durch Indexbildung im mathematischen Modus eine tiefgestellte kleine '$_1$' erreicht wurde.

Als Größe für die Einrückung wird der gleiche Wert wie beim Einzug zu Absatzanfängen verwendet (`\parindent`). Wird lokal etwas mehr Platz für solche Listen gewünscht, so kann `\parindent` lokal umdefiniert werden. Zur Ermittlung von bestimmten Textbreiten sei auf den Abschnitt 'Box-Register' verwiesen. Die Einrückungen sind nur erster und zweiter Stufe definiert, wünscht der Autor die Einrückung mal eine Stufe weiter, so hilft die folgende Definition:

```
\def\itemitemitem{\par\indent\indent
                  \hangindent3\parindent
                  \textindent}
```

Beispiele für eine Anwendung:

```
\leftline{Zuweisungen in \TeX}
\item{1.} Variablen-Zuweisung
\itemitem{a)} ‘integer’ Variable
\itemitem{b)} ‘dimen’ Variable
\itemitem{c)} ‘token’ Variable
\itemitem{}   $\vdots$
\item{2.} arithmetische Zuweisungen
\itemitem{a)} {\it advance}
\itemitemitem{a$_1$)} ‘integer’ Variable
\itemitemitem{a$_2$)} ‘dimen’ Variable
\itemitemitem{a$_3$)} ‘glue’ Variable
\itemitemitem{a$_4$)} ‘muglue’ Variable
\itemitem{b)} {\it multiply}
\itemitem{c)} {\it divide}
\par
```

liefert

Zuweisungen in TEX

1. Variablen-Zuweisung
   a) ‘integer’ Variable
   b) ‘dimen’ Variable
   c) ‘token’ Variable
   ⋮
2. arithmetische Zuweisungen
   a) *advance*
      a_1) ‘integer’ Variable
      a_2) ‘dimen’ Variable
      a_3) ‘glue’ Variable
      a_4) ‘muglue’ Variable
   b) *multiply*
   c) *divide*

Der Numerierungsteil wird jeweils rechtsbündig vor den Absatz gesetzt. Für bestimmte Aufzählungen ist eine linksbündige Aufzählung jedoch angenehmer. Auch hier kann man sich durch eine eigene Definition passender Makros leicht helfen. In Anlehnung an die Definition von `\item` und `\itemitem` sind die folgenden Festlegungen getroffen:

```
\def\litem{\par\noindent
               \hangindent=\parindent\ltextindent}
\def\litemitem{\par\noindent
               \hangindent=2\parindent\ltextindent}
\def\ltextindent#1{\hbox to \hangindent{#1\hss}\ignorespaces}
```

Anwendungsbeispiel:

```
{\parindent=4cm
\litem{\it Humane}
       Venezianische Renaissance-Antiqua  ---
       Diese Schrift ist hervorgegangen aus der
       humanistischen Minuskel des 15. Jahrhunderts.
       Die Serifen sind ein wenig ausgerundet. Die
       Achse der Rundungen ist nach links geneigt.
\litem{\it Garalde}
       Franz\"osische Renaissance-Antiqua ---
       Sie weist gr\"o\ss ere Unterschiede in der
       Strichdicke auf, der Querstrich des ‘e’ liegt
       waagerecht.
\litem{\it R\’eale}
       Barock-Antiqua --- Sie weist gr\"o\ss ere
       Unterschiede in den Strichdicken auf, die
       Rundungsachse steht fast senkrecht. Die Serifen
       sind bei Kleinbuchstaben unten waagerecht, oben
       schr\"ag angesetzt.
\par}
```

liefert

Humane	Venezianische Renaissance-Antiqua — Diese Schrift ist hervorgegangen aus der humanistischen Minuskel des 15. Jahrhunderts. Die Serifen sind ein wenig ausgerundet. Die Achse der Rundungen ist nach links geneigt.
Garalde	Französische Renaissance-Antiqua — Sie weist größere Unterschiede in der Strichdicke auf, der Querstrich des ‘e’ liegt waagerecht.
Réale	Barock-Antiqua — Sie weist größere Unterschiede in den Strichdicken auf, die Rundungsachse steht fast senkrecht. Die Serifen sind bei Kleinbuchstaben unten waagerecht, oben schräg angesetzt.

Verwandt mit der Ausgabeform des letzten Beispiel ist die gespiegelte Form, wobei die Markierungstexte jetzt rechts vom Text bündig am Rand stehen. Durch die abgewandelten Makros

```
\def\ritem{\par\noindent\hangindent=-\parindent
           \hangafter=0
           \rtextindent}
\def\ritemitem{\par\noindent
               \hangafter=0
               \hangindent=-2\parindent
               \rtextindent}
\def\rtextindent#1{\hbox to 0pt{\hskip\hsize
                     \hbox to 0pt{\hss#1}\hss}%
                   \ignorespaces}
```

wird bei der Anwendung

```
{\parindent=4cm
 \ritem{{\it Didone}}
       Klassizistische Antiqua ---
       Die Serifen sind waagerecht angesetzt.
       ...
```

folgende Ausgabe erreicht:

Klassizistische Antiqua — Die Serifen sind waagerecht angesetzt. Die Haar- und Grundstriche unterscheiden sich kräftig. Die Rundungsachse steht senkrecht.	*Didone*
Serifenbetonte Linear Antiqua — Haar- und Grundstriche unterscheiden sich kaum. Alle Schriften zeichnen sich durch die stark betonten Serifen aus.	*Mécane*
Serifenlose Linear-Antiqua — Diese Schriften werden häufig auch als "Grotesk" bezeichnet. Sie besitzen keine Serifen mehr.	*Lineale*

3.13 Seitenwechsel

Seitenwechsel geschehen im Normalfall vollautomatisch. Das TEX-Programm sucht entsprechend seines Umbruchalgorithmuses eine möglichst gute Stelle, um die einzelnen Seiten aufzuteilen. In der Praxis hat man jedoch oft spezielle Vorstellungen, ob eine bestimmte Umbruchstelle nun wirklich gut ist. Daher ist es sinnvoll, das Umbruchverfahren zu unterstützen, indem gute Trennpositionen mitangegeben werden. Jeden Seitenwechsel explizit setzen zu wollen, halte ich für unpraktisch, da bei Textänderungen dies eine Veränderung aller Angaben zur Folge hat.

Will man einen expliziten Seitenwechsel erzwingen, so gibt es hierfür den Befehl `\eject`. Ist die Seite an dieser Position aber noch nicht voll, so kommt es zu einer Fehlermeldung "underfull vbox ...". In vielen Fällen wird man diese ignorieren können. Lästig ist es dagegen, daß das TEX-Programm die bisher aufgelaufene Information für diese Seite dann gleichmäßig auf dieser Seite verteilt und so große Lücken zwischen einzelnen Textteilen bildet. Daher ist für einen expliziten Seitenwechsel die Befehlskombination `\vfill\eject` empfehlenswert. `\vfill` füllt dabei die Seite noch mit Leerplatz am Seitenende auf, falls dies nötig ist.

Besser ist es in jedem Fall, den Seitenumbruch durch Vorgabe guter Trennpositionen zu unterstützen. Insbesondere, wenn bestimmte Textfolgen zusammenhängend gedruckt werden sollen, bieten sich einige Befehle zur Markierung solcher Sequenzen an:

```
\goodbreak
\filbreak
\nobreak
```

Sie haben folgende Wirkung:

`\goodbreak`	markiert eine gute Umbruchstelle. Gleichzeitig geht damit auch ein Absatz zu Ende.

`\filbreak` läßt sich am besten durch folgende Beschreibung darlegen: "Beginne hier eine neue Seite, falls der bis zum nächsten `\filbreak` folgende Text nicht mehr auf die aktuelle Seite paßt!"

`\nobreak` verhindert einen Seitenumbruch an einer speziellen Stelle. Sinnvoll ist diese Angabe zum Beispiel zwischen einer Überschrift und dem darauf folgenden Text.
Die Angabe `\nobreak` funktioniert auch im normalen Fließtext, nur verhindert sie dort den Zeilenumbruch. Daher ist die `\nobreak`-Angabe nach dem Absatzende zu machen.

Eine andere Möglichkeit, den Seitenwechsel zu unterstützen, ist die Kombination, vertikalen Leerraum zu setzen und gleichzeitig

Umbruchstellen zu markieren. Die dafür verwendbaren Befehle sind:

```
\smallbreak     \medbreak     \bigbreak
```

Sie entsprechen zunächst in ihrer Wirkung den schon von vorher bekannten Befehlen `\smallskip`, `\medskip` und `\bigskip`. Gleichzeitig werden jedoch gute Umbruchpositionen markiert. Die 'Güte' der Trennstelle entspricht dabei dem Verhältnis von 1 : 2 : 4, also ist eine mit `\bigbreak` markierte Trennstelle viermal so gut wie eine durch `\smallbreak` gesetzte Position.

Zu beachten ist allerdings noch die folgende Feinheit. Diese drei Befehle entfernen unter Umständen Leerraum, der ihnen vorangeht. Dies geschieht immer dann, wenn der voranstehende Leerraum kleiner oder gleich dem zu setzenden ist. Also wirkt 2×`\medbreak` wie ein einzelnes `\medbreak`.
Im Gegensatz dazu wirkt ein `\medskip\medskip` wie `\bigskip`.

3.14 Seitennumerierung

Die Seitennummern werden durch das TEX-Programm, beginnend mit eins automatisch hochgezählt. Die dazu benutzte Zählvariable ist `\count0`, die normalerweise unter dem Namen `\pageno` besetzt wird. Umgestellt werden kann die Seitennummer zum Beispiel durch "`\pageno=73`". Die folgenden Seiten werden dann "74, 75 ..." numeriert. Mit Hilfe des `\advance`-Befehls kann auch inkremental fortgeschaltet werden:

```
\advance\pageno by 4
```

setzt die Seitennummer um 4 weiter. Referiert wird die Seitennummer durch den Befehl `\folio`. Dieser gibt die Seitennummer im Text aus. Hier kommt eine Besonderheit zum Tragen: Sind die Seitennummern *negativ* eingegeben worden, so wird automatisch rückwärts gezählt: '-1, -2, -3 ...'. Der Aufruf zur Ausgabe mit `\folio` erzeugt dann kleine römische Zahlen "i, ii, iii, iv, v ...".
Die Eingabe

```
\centerline{\it Wir befinden uns auf Seite \folio!}
```

erzeugt

Wir befinden uns auf Seite 46!

Die Seitennummer wird automatisch in der Fußzeile zentriert gesetzt. Im folgenden Abschnitt wird ausführlich diskutiert, auf welche Weisen das Layout für Seitennummer und Kopf- und Fußzeilen erfolgen kann.

3.15 Kolumnentitel: Seitenüberschriften, Seitenunterschriften

Das Standardlayout einer Ausgabeseite hat die Form, daß unter den Text zentriert die Seitennummer geschrieben wird, über die Seite wird 'nichts', das heißt eine leere Kopfzeile geschrieben. Das Aussehen von Kopf- und Fußzeilen wird durch die Befehle `\headline` und `\footline` definiert. Sie enthalten jeweils eine Folge von Befehlen, die den Inhalt der Ausgabezeilen bestimmen. Vorbesetzt sind diese mit

```
\headline={\hfil}
\footline={\hss\tenrm\folio\hss}
```

Wünscht man keine Seitennumerierung, so sagt man `\nopagenumbers`. Dies ist gleichbedeutend mit `\footline={\hfil}`. Der Befehl `\nopagenumbers` kann mit den Gruppenklammern '`{`' und '`}`' auch auf bestimmte Gültigkeitsbereiche eingeschränkt werden. Es wird im übrigen nur die *Ausgabe* unterdrückt, die Seitenzählung läuft weiter!

Der Kolumnentitel kann nun in vielfältiger Form gestaltet werden, wie die folgenden Beispiele zeigen. In der Praxis sollte man aber keine verspielten Darstellungsweisen für ernsthafte Arbeiten verwenden.
Bei den Beispielen werden einige Befehle verwendet, die bisher noch nicht erläutert wurden, zu diesen sei hier vorab eine Kurzdarstellung gegeben:

`\ifodd` prüft, ob die folgende Zahl *ungerade* ist.
`\else` leitet wie in normalen Programmiersprachen die Alternative zu einer `\if...`-Abfrage ab.
`\fi` beendet den Gültigkeitsbereich einer `\if...`-Abfrage.
`\vbox` setzt Bestandteile einer '`\vbox`' untereinander.
`\hss` bildet dynamischen (auch negativen) horizontalen Leerplatz.

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

19

Standardtitel: in der Mitte unten

```
\headline={\hfil}
\footline={\hss\tenrm\folio\hss}
```

19

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

Kolumnentitel: in der Mitte oben

```
\headline={\hss\tenrm\folio\hss}
\footline={\hss}
```

20

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

seitlich wechselnde Numerierung
— ohne Einzug

```
\headline={\tenrm
           \ifodd\pageno
             \hss\folio
           \else
             \folio\hss
           \fi}
\footline={\hss}
```

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

21

unten seitlich wechselnde Numerierung
— mit Einzug

```
\headline={\hss}
\footline={\tenrm
           \ifodd\pageno
              \hss\folio\quad
           \else
              \quad\folio\hss
           \fi}
```

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

∗ 21 ∗

Numerierung mit Schmuck
— unten Mitte

```
\headline={\hss}
\footline={\tenrm
           \hss
           $\ast$\ \folio\
           $\ast$\hss}
```

∗ Titeltext ∗

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

21

Letzter Kolumnentitel mit Leiste

```
\def\Buchtitel{Titeltext}
\headline={\vbox
   {\hrule
    \line{\strut
      $\ast$
      \hss\rm\Buchtitel\hss
      $\ast$}%
    \hrule\vss}}
\footline={\hss\tenrm\folio\hss}
```

40 A. B. Cäsario: Memoiren

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

Getrennt Verfasser und Werk linke Seite, Kapitelinhalt rechte Seite

```
\def\Buchtitel{Memoiren}
\def\Verfasser{A. B. C\"asario}
\def\Kapiteltitel{Jugendzeit}
\headline={\ifodd\pageno
    \Kapiteltitel\hss\folio
 \else   \folio\hss
    \Verfasser: \Buchtitel \fi}
\footline={\hss}
```

Jugendzeit 41

Spätestens bei Erstellung dieser Eingabe sind die verwendeten Makros zutiefst verschachtelt. Daher verwundert es den Autor bisweilen, wenn etwas fast auf Anhieb gelingt.

dito: nur folgende Seite

3.16 Einfügen von Illustrationen

Illustrationen werden nachträglich in für sie freigehaltenen Platz eingefügt. Zur Unterstützung dafür gibt es folgende Befehle:

```
\topinsert    ... vertikales Material ...   \endinsert
\midinsert    ... vertikales Material ...   \endinsert
\pageinsert   ... vertikales Material ...   \endinsert
```

Diese drei Befehle müssen jeweils immer als Paar mit dem Schlußbefehl `\endinsert` auftreten. Alles was zwischen ihnen angegeben wird, bleibt aufgehoben, bis es an die passende Stelle eingefügt werden kann. "`vertikales Material`" bedeutet, daß dies entweder ein vertikaler *skip*-Befehl oder eine '`vbox`' sein muß. Die Bedeutung der '`vbox`' wird im Abschnitt Box-Manöver näher erläutert.
Die Befehle haben im einzelnen folgende Wirkung:
`\topinsert` versucht, auf der aktuellen Seite oben so viel Platz zu halten, daß dieser für die gewünschte Information ausreicht. Ist zu wenig Raum vorhanden, so wird der betreffende Teil auf die folgende Seite gesetzt.
Im Beispiel

```
\topinsert
\vskip 6 true cm
\centerline{Abbildung 17a: Archidiskon Imperator}
\bigskip
\endinsert
```

wird zunächst unskaliert 6 cm Platz gelassen, etwa für ein Bild. Anschließend wird zentriert eine Bildbeschriftung ausgegeben, sowie noch etwas Leerraum nach unten gelassen.
`\midinsert` hat eine ähnliche Wirkung, nur wird zunächst versucht, den Platz an der aktuellen Stelle zu lassen. Ist nicht mehr genug Platz auf der Seite vorhanden, wird die Information an den Anfang der nächsten Seite gesetzt, wie bei `\topinsert`.
`\pageinsert` fügt eine vollständige Seite ein.

Einen Befehl '`\footinsert`' gibt es nicht, dies sind normale Fußnoten. Die Beschreibung dazu folgt direkt.

3.17 Fußnoten

Eine Fußnote[1] wird durch `\footnote{ }{ }` gesetzt. Das erste Klammerpaar umfaßt das Indexzeichen, das zweite den Fußnotentext. Hochgestellte kleine Nummern erreicht man durch Umschaltung in den *mathematischen Modus* und Setzen eines Superscripts, also: `\footnote{$^1$}{Siehe dazu ...}`.

Wird nur ein einfaches Zeichen als Indexsymbol, z.B. '7', benutzt, darf das erste Klammerpaar auch entfallen, also: `\footnote7{Dies ist die Bemerkung Nummer 7.}`

Mit Kenntnis der *plain*-TEX-Makros kann die Darstellung der Fußnoten auch variiert werden:

maximaler Fußnotenumfang

Durch TEX wird standardmäßig ein Parameter `\dimen\footins=8truein` gesetzt, der festlegt, daß der Umfang der Fußnoten auf einer Seite bis zu 8 Zoll ausmachen kann. (Der Gesamtseitenumfang beträgt übrigens 8,9 Zoll.) Will man nur etwa die Hälfte füllen, so kann durch `dimen\footins=4in` dies entsprechend reduziert werden.

Fußnotenlinie

Zu Beginn des Fußnotenteils wird eine Abgrenzungslinie gesetzt. Diese ist durch die Befehle

```
\def\footnoterule{\kern-3pt
                  \hrule width 2 true in
                  \kern 2.6pt}
```

definiert. Dies erzeugt eine 2 Zoll lange Trennlinie. Durch Ersetzung von "`2 true in`" durch "`\hsize`" kann die Linie zum Beispiel über die gesamte Seitenbreite gezogen werden. Will der Autor gar keine Trennlinie, so kann er einfach durch

```
\def\footnoterule{}
```

diese ganz abschalten.

Schriftwechsel

Zu beachten ist, daß der Fußnotentext immer in der Schrift gesetzt wird, die gerade beim Aufruf des `\footnote`-Befehls eingestellt war. Man sollte daher zu Beginn des Fußnotentextes immer die gewählte Schrift für den Fußnotentext einstellen.

[1] Siehe dazu auch "The TEXbook" Seite 116ff.

Häufig wird gewünscht, Fußnoten in einer kleineren Schrift mit einem geringeren Zeilenabstand zu setzen. Dazu sind einige Umstellungen erforderlich, so daß der einfachste Weg über eine eigene Makrodefinition führt. Diese seien hier angegeben, dabei jedoch auf das Kapitel über Makros verwiesen.

```
\font\eightrm=cmr8
\long\def\fussnote#1#2{{\baselineskip=9pt
     \setbox\strutbox=\hbox{\vrule height 7pt depth 2pt width 0pt}%
     \eightrm
     \footnote{#1}{#2}}}
```

Die Anwendung* des so definierten Befehls `\fussnote` ist am Ende dieser Seite zu sehen. Typischerweise werden die Fußnoten nur um einen oder zwei Schriftgrade verkleinert, sonst sind die Fußnoten nur sehr schwer lesbar.
Wird zusätzlich in der Definition die letzte Zeile zu

```
  \everypar{\hangindent=\parindent}%  zusätzlich
  \footnote{#1}{#2\everypar{}}}   %  verändert
```

erweitert, das heißt auch der `\hangindent` umgesetzt,** so wird der Fußnotenabsatz mit Einzug gesetzt. Andere Möglichkeiten, die hier aber nicht erläutert werden, sind automatische Numerierung, Zusatzregister für Fußnoten und ähnliches.

3.18 Trennen

Das TEX-Programm trennt automatisch. Es versucht, so wenig zu trennen wie möglich. Die Trennung wird im Normalfall nach einer amerikanischen Trenntabelle erfolgen. Diese ist für deutschsprachige Text nicht sehr geeignet. Es gibt jedoch auch deutsche Trenntabellen. Meist ist bei den verschiedenen Implementierungen dann ein anderer *"format-file"* zu laden. Die Angaben dazu sind allerdings von der Implementierung abhängig. Ich möchte zunächst davon ausgehen, daß eine deutsche Trenntabelle vorhanden ist.

Version 3: Mit der neuen TEX-Version gibt es für das Trennen einige sehr interessante Neuerungen. Die wichtigste ist das Trennen nach verschiedenen Sprachen. Die aktuelle Sprache wird durch das Register `\language` festgelegt. Typischerweise entspricht `\language=0` Englisch. Ob nun parallel beispielsweise `\language=1` tatsächlich deutsche Trennregeln enthält, hängt von der jeweiligen Implementierung ab. Theoretisch können gleichzeitig 256 verschiedene Trennregeln verwaltet werden. Erfreulicherweise ist es sogar möglich, im gleichen Absatz Teilstücke nach unterschiedlichen Trennregeln zu behandeln. Wird eine Sprache (`\language`) gesetzt, zu der keine Trennregeln geladen wurden, wird gar nicht getrennt.

Wörter, die vom TEX-Programm falsch oder ungünstig getrennt würden, können mit Trennvorgaben versehen werden. Ein Vortrenner hat die Form '`\-`', also zum Beispiel `Vor\-tren\-ner`. Ein mit Vortrennern versehenes Wort kann auch nur an den

* Für die Basisgröße 9 Punkt, das heißt nur etwas verkleinert, ist folgende Angabe zu empfehlen: Roman-Font 'cmr9' und als 'baselineskip' 11 pt, die Strutbox variiert zu "height 8pt depth 3pt width 0pt"

** Durch geringe Variationen können also die verschiedensten Formen erreicht werden. Automatisch Fußnoten mehrspaltig zu setzen, ist aber recht anstrengend.

angegebenen Positionen getrennt werden. Daneben gibt es noch die Möglichkeit, das Ausnahmelexikon um Wörter zu erweitern. Einen neuen Eintrag vollzieht man mittels des `\hyphenation`-Befehls. Durch `\hyphenation{Ur-instinkt PASCAL}` werden alle möglichen Trennungen auf Dauer festgelegt. Die Größe des Ausnahmelexikons ist allerdings beschränkt.

Probleme gibt es im Zusammenspiel zwischen Trennen und deutschen Umlauten. Deutsche Umlaute werden ja standardmäßig mittels `\"a`,`\"o`,`\"u` bzw. `\"A`,`\"O` und `\"U` eingegeben. Nach der Meinung des TEX-Programms ist dieser so gebildete Akzent etwas so besonderes, daß sofort mit dem Trennen aufgehört wird, falls in einem Wort ein Umlaut auftritt. Dies führt also dazu, daß diese Wörter nur bis zum ersten Umlaut getrennt werden. Damit nun Wörter wie "Öffentlichkeitsarbeitreferat" auch noch einigermaßen sauber verarbeitet werden, gibt es eine Notmaßnahme: die Umdefinition des `\"`-Befehls. Durch

```
\def\"#1{{\accent"7F #1\penalty10000\hskip 0pt plus 0pt}}
```

wird die Wirkung erreicht, daß ein Wort nach einem Umlaut zu Ende geht, ein neues Wort beginnt, das dann auch wieder getrennt werden darf. Dabei wird aber noch durch `\penalty10000` verhindert, daß zwischen den beiden so geschaffenen Teilwörtern ein Zeilenumbruch erfolgt. Die Trennstellen im letzten Beispielwort würden also praktisch im Restwort "ffentlichkeitsarbeitreferat" gesucht, wo es nun auch noch eine Menge gibt. Der geneigte Leser möge das letzte Makro als Hilfsmittel nehmen und diese Unhandlichkeit übersehen.

Das zweite Problem beim Trennen in der deutschen Sprache ist die Tatsache, daß eine Reihe von Wörtern durch das Trennen ihre Schreibweise verändern:
Aus 'backen' wird 'bak–ken', aus 'Brennessel' wird 'Brenn-nes-sel'. Einen Ausweg bietet der `\discretionary`-Befehl, der allerdings nur in 'Handarbeit' bei den einzelnen Wörtern zu nutzen ist.

```
\discretionary{ <Text vor der Trennung > }
              { <Text nach der Trennung> }
              {   <Text ohne Trennung >  }
```

Am leichtesten ist dies in der Anwendung zu sehen. Durch

```
\def\ck{\discretionary{k-}{k}{ck}}
\def\ff{ff\discretionary{-}{f}{}}
\def\nn{nn\discretionary{-}{n}{}}
```

werden einige Befehle definiert, die in der Anwendung auf

```
ba\ck en   Schi\ff ahrt   Bre\nn essel
```

im Fall der Trennung zu orthographisch richtigen Ergebnissen führen.

Will man wissen, wie bestimmte Wörter getrennt werden, kann durch den Befehl `\showhyphens{ testworte ... }` ein Protokoll der möglichen Trennpositionen erreicht werden.

Intern wird das Trennen über den Zeilenumbruch abgewickelt, der für alle 'Unschönheiten', dazu gehört auch das Trennen, Minuspunkte summiert. Dieses Verfahren wählt dann die Lösung, die am wenigsten schlecht ausgefallen ist.
Setzt man `\hyphenpenalty=10000`, so wird kaum noch getrennt. Weitere Minuspunkte gibt es in diesem Zusammenhang für zwei aufeinanderfolgende Trennungen (`\doublehyphendemerits=10000`), für das Trennen in der vorletzten Zeile eines Absatzes (`\finalhyphendemerits=5000`) oder in letzten Zeile einer Seite (`\brokenpenalty=100`). Wobei die letzteren durch das Seitenumbruchverfahren ermittelt werden. Und zum guten Schluß noch einen Parameter: Wird `\pretolerance=10000` gesetzt, so wird in keinem Fall mehr getrennt. Der Test auf mögliche Trennungen unterbleibt dann vollständig.

3.19 Waagerechte und senkrechte Striche

Waagerechte und senkrechte Striche werden im TEX mittels `\hrule` und `\vrule` gezogen. Schreibt man `\hrule` mitten in einem Text, so wird der Paragraph beendet und ein waagrechter Strich über die ganze Seite gezogen. Die Strichdicke ist mit 0.4 pt voreingestellt. `\hrule` und `\vrule` besitzen 3 optionale weitere Angaben, die das Aussehen des Strichs definieren. Ein 'Strich' im Sinne von TEX ist nichts weiter als ein schwarzer Kasten. Zum Beispiel dieser Kasten '▮' ist entstanden aus:

```
\vrule height 4pt width 3pt depth 2pt
```

`\vrule` und `\hrule` besitzen die gleichen Zusatzangaben, jedoch sind diese unterschiedlich vorbesetzt:

		`\hrule`	`\vrule`
Breite	`width`	$*$	0.4 pt
Höhe	`height`	0.4 pt	$*$
Tiefe	`depth`	0.0 pt	$*$

'$*$' bedeutet, daß die Größe vom Kontext abhängt. Für `\hrule` ist `width` gleich der Breite der Umgebung. Dies ist im Normalfall zwischen zwei Absätzen die Zeilenlänge `\hsize`. Dagegen ermitteln sich die Werte von `\vrule` aus der *vertikalen box*, in der sich die `\vrule` befindet.

Die Anwendung wird von folgenden Regeln geleitet: `\hrule` beendet einen Absatz und erzeugt vom linken Rand aus einen neuen Strich. `\vrule` kann auch innerhalb einer Zeile angewendet werden und wird im Umbruchmechanismus für den Absatz berücksichtigt. Ob ein Strich — `\hrule` oder `\vrule` — nun waagrecht oder senkrecht *aussieht*, hängt von den Werten ab, die man zusätzlich angibt. Dieser Strich '▬', wurde mit

```
\vrule width 1 true in height 0.5 true cm depth 0pt
```

produziert. ('in' ist die Einheit für *inch*.) Die Linie liegt auf der Grundlinie der Zeile. Soll sie etwas höher kommen, ist bei gleicher Dicke folgendes anzugeben:

```
\vrule width 1 true in height 0.75 true cm depth -0.25 truecm
```

Dann erhält man ‘ ’. Die effektive Dicke ist also die Summe von `height` und `depth`. Man beachte in folgendem Beispiel besonders, daß kein zusätzlicher Leerraum erzeugt wird und daß `\hrule` einen Absatz beendet:

```
Man erh\"alt \hrule width 3 cm und weiter \hrule width 3cm und weiter
```

Man erhält
und weiter

Die typischen Anwendungen von `\hrule` und `\vrule` sind in Kombination mit etwas Leerplatz zu finden:

```
\hrule \smallskip
\leftline{Neue Ergebnisse: 1 2 3 4 5 6 7 8 9 10}
\smallskip \hrule
```

ergibt

Neue Ergebnisse: 1 2 3 4 5 6 7 8 9 10

Noch ein Beispiel:

```
\centerline{\vrule height 4pt width 6cm}
\medskip
\centerline{\bf NOCH KEIN ENDE !}
\medskip
\centerline{\vrule height 4pt width 6cm}
```

liefert

NOCH KEIN ENDE !

Wie man Kästchen und Rahmungen erhält, ist im Kapitel ‘Wie TEX arbeitet’ erläutert.

3.20 Fehlermarkierungen

Zeilen (oder die später ausführlich erläuterten Boxen), die überfüllt sind, werden am Zeilenende durch einen Fehlerbalken markiert. Da dies bei geringfügigen Überschreitungen, die sonst gar nicht mehr auffielen, nicht wünschenswert ist, wird durch

```
\overfullrule=0pt
```

die Breite des Fehlerbalkens auf Null gesetzt, damit verschwindet er ganz.

Auf der anderen Seite kann für geringfügige Abweichungen die Fehlerschranke erhöht werden, ab der eine Meldung erfolgt. Vorbesetzt ist der Parameter `\hfuzz` mit `\hfuzz=0.1pt`, also ein sehr geringer Wert. Bewährt hat sich die Angabe `\hfuzz=1 pt`, damit werden Überfüllungen bis zu 1/3 Millimeter weder protokolliert noch durch eine Fehlermarkierung in der Ausgabe gekennzeichnet.

Den entsprechenden Parameter gibt es auch für überfüllte Seiten (bzw. ‘vboxen’): `\vfuzz`. Er hat die entsprechende Wirkung. æ

4 Schriftenkatalog

4.1 Schriftfamilien

Werden in einem Text *verschiedene* Schriften benutzt, so sind dies immer Schriften einer *Schriftfamilie*. Eine Schrift hat immer bestimmte charakteristische Merkmale, zum Beispiel Größenverhältnisse zwischen Ober- und Unterlängen, Stärke der Serifenausprägung*. Die typischen Variationen der Schrift einer *Schriftfamilie* sind zunächst die **fette (boldface)** Schrift, die *italic*-Schrift oder *Kursive* und die *slanted*-Schrift. Dabei besteht die *slanted*-Schrift lediglich aus *schräggestellten* Buchstaben, in *italic* haben diese auch ein anderes Aussehen. Die Serifen werden anders gestaltet.

Werden Schriften nun in verschiedenen Größen ausgegeben — der Setzer nutzt als Einheit immer das Maß *Punkt (pt)* — so wird das Bild eines Buchstabens nicht einfach vergrößert. Stattdessen werden unter anderem die Verhältnisse der Strichdicken zur Buchstabenhöhe variiert, um ein ästhetisch gutes Aussehen zu erreichen.

Diese Schriften werden dann in ihren Design-Größen 5 pt, 6 pt, 7 pt usw. verwendet. Die Buchstaben einer kleineren Schrift, etwa 5 pt hoch, sind zwar nur halb so hoch, wie die entsprechende 10 pt-Schrift, aber breiter als eine nur linear verkleinerte Schrift. Dies geschieht, um sie besser lesbar zu gestalten.

10 pt-Schrift: Typographie ist eine Kunst.
5 pt-Schrift: Typographie ist eine Kunst.

4.2 Schriftanwahl

Die Schriften des TEX-Systems sind die im Rahmen des TEX-Projekts entwickelten Schriften der “computer modern”-Schriftfamilie.† Die Anwahl der Standardschriften

* Serifen sind die ‘Häkchen’ an den Buchstabenenden. Dies ist eine serifenfreie Schrift.

† Diese Schriften sind im Band E der “computer & typesetting”-Serie von Donald E. Knuth beschrieben.

ist vordefiniert und kann durch folgende Befehle benutzt werden:

`\rm`	wählt "roman" (Basisschrift) an:	Zu lesen bedeutet, Aussagen von Worten zu erfahren. Lesbarkeit heißt, dies schnell, leicht und sicher zu tun.
`\bf`	wählt "**boldface / fett**" an:	**Zu lesen bedeutet, Aussagen von Worten zu erfahren. Lesbarkeit heißt, dies schnell, leicht und sicher zu tun.**
`\it`	wählt "*italic (kursiv)*" an:	*Zu lesen bedeutet, Aussagen von Worten zu erfahren. Lesbarkeit heißt, dies schnell, leicht und sicher zu tun.*
`\sl`	wählt "*slanted*" an:	*Zu lesen bedeutet, Aussagen von Worten zu erfahren. Lesbarkeit heißt, dies schnell, leicht und sicher zu tun.*
`\tt`	wählt "`typewriter`" an:	`Zu lesen bedeutet, Aussagen von Worten zu erfahren. Lesbarkeit heißt, dies schnell, leicht und sicher zu tun.`

Innerhalb von mathematischen Formeln haben diese Befehle eine besondere Wirkung: Dort wird nicht nur die Schrift gewechselt, sondern gleichzeitig werden auch die Schriften für Exponenten und Indizes geändert. Genau betrachtet bedeuten die Befehle `\rm` bis `\tt` den Wechsel in eine andere *Schriftfamilie*.

Dagegen bewirken die Befehle `\tenrm`, `\tenbf`, `\tenit`, `\tensl` und `\tentt` nicht einen Wechsel der Schriftfamilie, sondern nur die Einstellung einer einzelnen Schrift. Wird dabei zum Beispiel die Schrift `\tenit` umdefiniert, so wird bei einem Wechsel der Schriftfamilie durch `\it` auch die neue `\tenit` mitangewählt.
Die kleineren Schriftgrade sind unter den Befehlen

```
\sevenrm          \fiverm
\sevenbf          \fivebf
```

vordefiniert. Automatisch werden diese Größen für Exponenten und Indizes erster und zweiter Stufe verwendet. Es sei aber noch darauf hingewiesen, daß bei der Anwahl einer kleineren Schrift immer noch der alte Zeilenabstand erhalten bleibt. Diesen sollte man geeignet mitwechseln. Kleine vordefinierte Formen der Standardschriften:

- Durch `\sevenrm` wird in sieben-punkt roman geschaltet.
- Durch `\fiverm` wird in fünf-punkt roman geschaltet.
- **Durch `\sevenbf` wird in sieben-punkt boldface geschaltet.**
- **Durch `\fivebf` wird in fünf-punkt boldface geschaltet.**

Dies sind die Standardschriften. Neben diesen Fonts stehen noch eine Reihe anderer Schriften zur Verfügung, die allerdings zunächst über den `\font`-Befehl benannt werden müssen. Dazu ist unten mehr zu lesen.

Will man vollständig portable TEX-Eingaben erzeugen, sollten nur die Standardschriften verwendet werden.

4.3 Vergrößerung — global

Vergrößerungen der Schriften können in auf zwei Arten gemacht werden:

Auf der einen Seite durch globale Umsetzung der *magnification* auf einen anderen Wert, aber dann werden alle Dinge *außer* der Papiergröße vergrößert. Dies bewirkt, daß das gesamte Dokument in einer anderen Vergrößerung ausgegeben wird. Dieser Text wurde zum Beispiel in `\magstep0` gesetzt. Das entspricht einer Basisschrift in der Größe von 10pt. Als Werte für die Vergrößerung sind folgende Angaben erlaubt:

Angabe `\magnification=`	Faktor
`\magstep0`	1.000
`\magstephalf`	1.095
`\magstep1`	1.200
`\magstep2`	1.440
`\magstep3`	1.728
`\magstep4`	2.074
`\magstep5`	2.488

Damit wird ein *globaler* Vergrößerungsfaktor gesetzt. *Dies kann auch nur einmal am Anfang gemacht werden.* Weitere Versuche, `\magnification` zu ändern, führen zu Fehlermeldungen. Also ist nur am Dokumentanfang etwa `\magnification=\magstep1` zu setzen. Sinnvoll ist dies insbesondere bei späterer verkleinerter Reproduktion.

4.4 Fonts in eigenen Vergrößerungen

Daneben gibt es noch die Möglichkeit, eine vorhandene Schrift mit einem spezifischen Vergrößerungsfaktor aufzurufen. Dazu wird eine Schriftbezeichnung vollzogen. Der Befehl hierfür ist `\font`. Zum Beispiel

```
\font\meinfont=cmbx10 scaled \magstep2
```

definiert den vergrößerten boldface-Font unter dem Befehlsnamen `\meinfont`.

Das ist jetzt in Meinfont.

Der Aufruf dazu ist:

```
{\meinfont Das ist jetzt in Meinfont.}
```

Neben dem Schlüsselwort `scaled` kann auch mit Hilfe von `at` vergrößert werden. Der dem obigen Beispiel äquivalente Befehl heißt

```
\font\meinfont=cmbx10 at 14.4pt
```

Damit wird nicht mit einem Skalierungsfaktor, sondern mit der direkten Größenangabe gearbeitet. Der bei `scaled` nötige Faktor ergibt sich aus dem Quotienten von $(14.4\,\mathrm{pt}/10\,\mathrm{pt}) * 1000$. Skalierungsfaktoren werden im TEX-Programm grundsätzlich in Promille-Einheiten angegeben.

Allerdings ist mit der Angabe `\magstep2` aus dem `\font`-Befehl noch nicht die herauskommende Vergrößerung festgelegt. Dieses `\magstep2` wird nämlich noch mit der

globalen Vergößerung `\magnification` multipliziert! War der globale Vergrößerungsfaktor 1.0, dies entspricht `\magstep0`, ändert sich nichts, war er aber zum Beispiel 1.2, dies entspricht `\magstep1`, dann ist die effektive Größe `\magstep3`.

Einen Vorteil bietet die `at`-Angabetechnik: Mit Hilfe von "`true`" bei der Angabe `\font\meinfont=cmbx10 at 14.4truept` kann die Skalierung durch `\magnification` unterdrückt werden.

Alle diese Vergrößerungstechniken bewirken, daß Schriften verwendet werden, die für besser auflösende Geräte gedacht sind, bei denen einfach eine lineare Vergrößerung stattgefunden hat. Wird eine 12pt hohe Schrift benötigt, sollte auch etwa "cmr12" verwendet werden und nicht die vergrößerte "cmr10".

Allerdings muß für Sonderschriften stets geprüft werden, ob die Schrift tatsächlich auch existiert.

4.5 Vorhandene Schriften

Tabelle der Standardschriften, die eine Minimalausstattung bilden:

roman		*mathematische Symbole*	
`\tenrm` (`\rm`)	= cmr10	`\tensy`	= cmsy10
`\sevenrm`	= cmr7	`\sevensy`	= cmsy7
`\fiverm`	= cmr5	`\fivesy`	= cmsy5
bold extended		*mathematischer Text*	
`\tenbf` (`\bf`)	= cmbx10	`\teni` (`\mit`)	= cmmi10
`\sevenbf`	= cmbx7	`\seveni`	= cmmi7
`\fivebf`	= cmbx5	`\fivei`	= cmmi5
slanted		*große mathematische Symbole*	
`\tensl` (`\sl`)	= cmsl10	`\tenex`	= cmex10
italic			
`\tenit` (`\it`)	= cmti10		
typewriter type			
`\tentt` (`\tt`)			

Welche weiteren Schriften noch tatsächliche benutzt werden können, hängt von der Implementierung und dem vorhandenen Fontvorrat ab.

In einigen älteren Implementierungen sind auch noch die älteren Versionen der "computer modern"-Schriftfamilie in Benutzung, dort beginnt der Name der jeweiligen Schrift nicht mit 'c', sondern mit 'a' — für "almost computer modern". Die älteren und neuen Fonts unterscheiden sich zum Teil in den Laufweiten der einzelnen Zeichen, so daß es beim Wechsel von der alten zur neuen Version zu Änderungen im Umbruch kommen kann.

4.6 "Computer Modern" Schriften

Die folgenden Schriftbeispiele führen jeweils eine Originalgröße und die um Faktor 2,488 (`\magstep5`) vergrößerte Form an. Es werden nur die Textschriften einer Vollimplementierung aller 'computer modern' Schriften aufgeführt, wie sie zur Zeit als Eingabe für METAFONT erhältlich sind.

cmb10 — bold **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx12 — bold extended **Typographie ist eine Kunst.**

Typographie ist Kunst.

cmbx10 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx9 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx8 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx7 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx6 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbx5 — bold extended **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmbxsl10 — bold extended slanted ***Typographie ist eine Kunst.***

Typographie ist eine Kunst.

cmbxti10 — bold extended italic ***Typographie ist eine Kunst.***

Typographie ist eine Kunst.

cmcsc10 — caps and small caps Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmdunh10 — dunhill roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmff10 — funny roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmfib8 — fibonacci roman **Typographie ist eine Kunst.**

Typographie ist eine Kunst.

cmr17 — roman Typographie ist eine Kunst.

Typographie

cmr12 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr10 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr9 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr8 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr7 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr6 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmr5 — roman Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmsl12 — slanted *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmsl10 — slanted *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmsl9 — slanted *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmsl8 — slanted *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmss17 — sans serife

Typographie ist eine Kunst.

Typographie

cmss12 — sans serife

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmss10 — sans serife

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmss9 — sans serife

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmss8 — sans serife

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmssbx10 — sans serife bold extended

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmssdc10 — sans serife demibold

Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmssi17 — sans serife italic

Typographie ist eine Kunst.

Typographie

cmssi12 — sans serife italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmssi10 — sans serife italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmssi9 — sans serife italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmssi8 — sans serife italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmssq8 — sans quotation Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmssqi8 — sans quotation slanted *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmtcsc10 — typewriter caps & small caps Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmti12 — text italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmti10 — text italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmti9 — text italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmti8 — text italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmti7 — text italic *Typographie ist eine Kunst.*

Typographie ist eine Kunst.

cmtt12 — typewriter type `Typographie ist eine Kunst.`

`Typographie ist Kunst.`

cmtt10 — typewriter type `Typographie ist eine Kunst.`

`Typographie ist eine Kunst.`

cmtt9 — typewriter type `Typographie ist eine Kunst.`

`Typographie ist eine Kunst.`

cmtt8 — typewriter type `Typographie ist eine Kunst.`

`Typographie ist eine Kunst.`

cmu10 — unslanted italic Typographie ist eine Kunst.

Typographie ist eine Kunst.

cmvtt10 — variable width typewriter `Typographie ist eine Kunst.`

`Typographie ist eine Kunst.`

5 Mathematischer Formelsatz

5.1 Vorbemerkung

Der Satz mathematischer Formeln wurde im Setzergewerbe immer als eine Strafarbeit betrachtet — aus gutem Grund. Soll ein mathematischer Ausdruck auch ästhetisch ansprechend gesetzt sein, so ist mindestens ein minimales Verständnis beim Setzer Voraussetzung. Der Satzaufwand war stets enorm hoch, da mathematischer Formelsatz immer eine Art von Tabellensatz ist.

Im TEX-System ist jedoch der Eingabeaufwand gerade für mathematische Formeln extrem niedrig. Die Erfahrung aus mehrjährigen praktischen Anwendungen hat gezeigt, daß die Angst vor der Eingabe der Mathematikteile einer Arbeit völlig unbegründet war, dies ging meist sehr einfach. Ein Grund für die einfache Verwendung des Formelsatzes liegt im standardisierten Layout mathematischer Ausdrücke, so daß hierbei kaum Layoutvorschriften anzugeben sind.

5.2 Grundsätzliches

Einige wenige Grundsätze sind bei der Eingabe mathematischer Formeln von gleichbleibender Bedeutung. Wenn diese beachtet werden, so sind viele Dinge von vornherein klar.

— **Gruppenklammern**
Stets müssen zusammengehörende Teile mit '`{`' und '`}`' gruppiert werden, damit sie richtig gesetzt werden. Beispielsweise ergibt `$x_ij$` den Ausdruck x_ij, wobei aber `$x_{ij}$` als x_{ij} gemeint war. Der Indexteil zu x bestand also aus beiden Zeichen `ij`. Um dieses auszudrücken, muß man sie mit '`{}`' in Klammern setzen.

— **'oben' und 'unten'**
Stets werden Formelteile, die *unten* gesetzt werden sollen, mit '`_`' (Unterstrich) eingeleitet und Teile, die nach oben gehören, mit '`^`' (Dach/Zircumflex).

— **Unterscheidung von optischem Bild und Funktion**
Gerade in der Mathematik ist die Unterscheidung der Funktion eines Symbols zu seiner Umgebung als Operator, Relation oder ähnlichem wichtig. Jeder Befehl gibt nicht nur das Drucksymbol, sondern gleichzeitig auch eine mathematische Funktion an. Daher gibt es eine Reihe von Zeichen mit gleichem Aussehen, aber

unterschiedlichem Satzverhalten, die unter verschiedenen Namen referiert werden. Die Abstände zwischen den Formelelementen werden dabei nach der Funktion des einzelnen Bestandteils geregelt. Es wird zum Beispiel zwischen Operatoren wie $\sum$, $\int$ und Relationen wie '=', '$\leq$' und binären Operatoren wie '+', '$*$' und '$-$' unterschieden.

— **Begrenzung einer Formel**
Der Beginn einer Formel wird durch das `$`-Symbol eingeleitet. Damit geht TEX in den *mathematischen Satzmodus* über und bleibt bis zum nächsten '`$`' darin.
Werden zwei Dollarzeichen '`$$`' zum Beginn und zum Ende einer Formel eingegeben, so wird diese mathematische Formel als eigene Textzeile in hervorgehobener Weise gesetzt. Auch die Positionierung von Indexzeichen etc. verändert sich dann.

— **Schriftarten**
Im Formelsatz werden alle Buchstaben in der Schriftart *math-italic* gesetzt, die übrigen Zeichen normalerweise in der Schriftform "roman".
Durch Angabe von `\rm`, `\bf` und `\it` können andere Schriften gewählt werden.

— **Struktierung der Eingabe**
Die Leerstellen in der Eingabe haben *keine* Wirkung für die Abstände der Formelelemente. Lediglich als Trennzeichen zwischen Befehl und Symbolen werden sie benutzt, zum Beispiel in `\sin x`. Die Abstände werden durch TEX bestimmt. Um die Eingabe besser zu verstehen und auch später noch einmal nachvollziehen zu können, sollten Sie allerdings geeignet Leerzeichen miteingeben.

Die Berücksichtigung dieser wenigen Grundsätze erleichtert den Satz mathematischer Formeln.

5.3 Griechische Buchstaben

Die Eingabe griechischer Buchstaben geschieht einfach über den üblichen Namen des Buchstaben. Befehle für kleine griechische Buchstaben beginnen mit einem Kleinbuchstaben am Befehlsanfang, für große mit einem Großbuchstaben. Das griechische Alphabet wird also entsprechend folgender Tabelle eingegeben:

α	`\alpha`	ι	`\iota`	ϱ	`\varrho`
β	`\beta`	κ	`\kappa`	σ	`\sigma`
γ	`\gamma`	λ	`\lambda`	ς	`\varsigma`
δ	`\delta`	μ	`\mu`	τ	`\tau`
ϵ	`\epsilon`	ν	`\nu`	υ	`\upsilon`
ε	`\varepsilon`	ξ	`\xi`	ϕ	`\phi`
ζ	`\zeta`	o	`o`	φ	`\varphi`
η	`\eta`	π	`\pi`	χ	`\chi`
θ	`\theta`	ϖ	`\varpi`	ψ	`\psi`
ϑ	`\vartheta`	ρ	`\rho`	ω	`\omega`

`\omicron` gibt es nicht, da es wie "o" aussieht. Mittels `\def\omicron{o}` kann man sich der Vollständigkeit halber eins machen. Einige Buchstaben besitzen Ausgabevarianten; dort ist der zweiten Version ein "var" im Namen vorangestellt: so etwa `\pi` und `\varpi`.

Große griechische Buchstaben:

A	{\rm A}	I	{\rm I}	Σ	\Sigma
B	{\rm B}	K	{\rm K}	T	{\rm T}
Γ	\Gamma	Λ	\Lambda	Υ	\Upsilon
Δ	\Delta	M	{\rm M}	Φ	\Phi
E	{\rm E}	N	{\rm N}	X	{\rm X}
Z	{\rm Z}	Ξ	\Xi	Ψ	\Psi
H	{\rm H}	Π	\Pi	Ω	\Omega
Θ	\Theta	P	{\rm P}		

Für griechische Zeichen, die in der Normalschrift identisch aussehen, sind *keine* speziellen Befehle vorhanden. Dort reicht ja die einfache Angabe des normalen Buchstabens, versehen mit der Schriftanwahl '`\rm`'. Ohne diese Angabe werden Buchstaben im Mathematiksatz standardmäßig in der Schrift *math italic* gesetzt. Normalerweise werden die griechischen Buchstaben in der Schriftart 'roman' gesetzt. Es gibt sie aber noch in weiteren Schriftvarianten als 'italic'- und 'bold'-Fassung; griechische Kleinbuchstaben sind nur in der 'italic' Fassung vorhanden.

in italic:

A	A	I	I	Σ	{\it\Sigma}
B	B	K	K	T	T
Γ	{\it\Gamma}	Λ	{\it\Lambda}	Υ	{\it\Upsilon}
Δ	{\it\Delta}	M	M	Φ	{\it\Phi}
E	E	N	N	X	X
Z	Z	Ξ	{\it\Xi}	Ψ	{\it\Psi}
H	H	Π	{\it\Pi}	Ω	{\it\Omega}
Θ	{\it\Theta}	P	P		

in 'bold':

A	{\bf A}	**I**	{\bf I}	$\boldsymbol{\Sigma}$	{\bf\Sigma}
B	{\bf B}	**K**	{\bf K}	**T**	{\bf T}
$\boldsymbol{\Gamma}$	{\bf\Gamma}	$\boldsymbol{\Lambda}$	{\bf\Lambda}	$\boldsymbol{\Upsilon}$	{\bf\Upsilon}
$\boldsymbol{\Delta}$	{\bf\Delta}	**M**	{\bf M}	$\boldsymbol{\Phi}$	{\bf\Phi}
E	{\bf E}	**N**	{\bf N}	**X**	{\bf X}
Z	{\bf Z}	$\boldsymbol{\Xi}$	{\bf\Xi}	$\boldsymbol{\Psi}$	{\bf\Psi}
H	{\bf H}	$\boldsymbol{\Pi}$	{\bf\Pi}	$\boldsymbol{\Omega}$	{\bf\Omega}
$\boldsymbol{\Theta}$	{\bf\Theta}	**P**	{\bf P}		

Die weiteren Varianten wären *typewriter type* und *slanted*. Sie sind mittels `\tt` und `\sl` anwählbar. Die griechischen Buchstaben sind allerdings nur Symbole, die für die Benutzung im mathematischen Satz gestaltet wurden. Sie dienen nicht dem Satz eines griechischen Textes. Dazu benötigt man spezielle griechische Schriften.

Dies war nur zur Einstimmung. Die Beschreibung aller standardmäßig vorhandenen Symbole folgt in weiteren Abschnitten. Wie man mathematische Formeln in verschiedenen Satzvarianten setzt, ist in Kapitel 9 "Variation des Formelsatzes" ausführlich beschrieben. Nur noch eine Anmerkung: Soll ein mathematisches Symbol im normalen Text benutzt werden, muß man mit "`$ ... $`" zwischendurch in den mathematischen Modus wechseln.

5.4 Exponenten und Indizes

Exponenten und Indizes werden mit den beiden Zeichen '`^`' und '`_`' gebildet. Dabei ist zu beachten, daß diese nur auf das folgende Zeichen bzw. auf die folgende durch '`{`' und '`}`' gebildete Gruppe wirken.

`$x^2-y$` und `$x^{2-y}$` *ergeben* $x^2 - y$ und x^{2-y}.

Indizes und Exponenten höherer Ordnung werden selbstverständlich automatisch kleiner gesetzt. Exponenten und Indizes gibt es in zwei Größenabstufungen, jeweils erster und zweiter Stufe. Sind in der Formel noch tiefer verschachtelte Strukturen vorhanden, so werden die Exponenten und Indizes aber nicht weiter verkleinert. Wenn Exponenten oder Indizes erster Stufe gesetzt werden, befindet sich das TEX-Programm im sogenannten *scriptstyle*, beim Satz in zweiter Stufe im *scriptscriptstyle*. Durch die Befehle `\scriptstyle` und `\scriptscriptstyle` kann auch ausdrücklich dafür gesorgt werden, daß die Information so gesetzt wird, als wenn gerade ein Index- oder Exponentausdruck erster bzw. zweiter Stufe gesetzt werden sollte.

Die folgenden Beispiele verdeutlichen die Verwendung von Exponent- und Indexausdrücken.

`$x^2y^2$`	x^2y^2
`$x ^ 2y ^2 $`	x^2y^2
`$x_2y_2$`	x_2y_2
`$_2F_3$`	${}_2F_3$
`$x^{2y}$`	x^{2y}
`$x^{2^x}$`	x^{2^x}
`$x^{2^{2^x}}$`	$x^{2^{2^x}}$
`$K_n^+,K_n^-$`	K_n^+, K_n^-
`$y_{x^2}$`	y_{x^2}
`$z^*_{ij}$`	z^*_{ij}
aber	
`${z^*}_{ij}$`	${z^*}_{ij}$

Im folgenden kommt eine Feinheit zu tragen. Die Stellung von Index und Exponent hängt von dem Term ab, zu dem diese gehören. Im folgenden Beispiel gehören die Exponenten '2', '3' und '4' jeweils zu dem vorangehenden 'x' bzw. der vorangehenden Klammer ')'.

`$((x^2)^3)^4$`	$((x^2)^3)^4$

Im Gegensatz dazu werden durch explizite Klammerbildung Subformeln erzeugt. Mit diesen werden dann die Exponent- und Indexpositionen bestimmt. Dann geht nicht

nur die Größe des letzten Zeichens in die Positionierung ein, sondern die Größe der kompletten Unterstruktur:

`${{{x^2}^3}^4$` $((x^2)^3)^4$

Ein besonders gebräuchlicher 'Exponent' ist das Zeichen für die Ableitung '′'. Der TEX-Befehl dazu ist `\prime`.

`$y_1^\prime$` y_1'

An Stelle der Befehlsfolge `^\prime` darf man auch ein Apostroph eingeben:

`$f''[g(x)]g'(x)$` $f''[g(x)]g'(x)$

Exponenten lassen sich im normalen Textsatz als Fußnotenkennzeichnung verwenden, zum Beispiel durch "`\footnote{$^7$}{ .. text ..}`".

5.5 Wurzelzeichen

Das Wurzelzeichen wird mit `\sqrt` eingegeben, die n-te Wurzel mit `\root n \of`. Dabei wird die Größe des Wurzelzeichens automatisch bestimmt. Soll mehr als ein Zeichen unter dem Wurzelstrich stehen, sind wie üblich Klammern zu setzen.

`$\sqrt x$` $\sqrt{x}$

`$\sqrt{x^3 + \sqrt\alpha}$` $\sqrt{x^3 + \sqrt{\alpha}}$

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

`$\root 3 \of {1 + \sqrt\alpha}$` $\sqrt[3]{1 + \sqrt{\alpha}}$

Und — weil schon arg groß — jetzt mal im *display-style*:

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

liefert

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

Anmerkung: Das Wurzelzeichen wird aus mehreren Teilzeichen zusammengesetzt. Die größeren Zeichen besitzen senkrechte Aufstriche, die aus mehreren kurzen Teilstrichen bestehen. So ist es möglich, beliebig große Wurzeln zu erzeugen.

5.6 Kennzeichnung durch Akzente und Striche

Die Befehle `\overline` und `\underline` bewirken waagerechte Striche über beziehungsweise unter einem Ausdruck.

Beispiel: `$\overline{m+n}$` ergibt $\overline{m+n}$ und `$\underline{m*n}$` ergibt $\underline{m*n}$.

Akzente seien am Beispiel von "x" gezeigt.

`$\hat x$` $\hat{x}$

`$\check x$` $\check{x}$

`$\tilde x$` $\tilde{x}$

`$\acute x$` $\acute{x}$

`$\grave x$` $\grave{x}$

`$\dot x$` $\dot{x}$

`$\ddot x$` $\ddot{x}$

`$\breve x$`	$\breve{x}$
`$\bar x$`	$\bar{x}$
`$\vec x$`	$\vec{x}$

Längere Tilden und Dächer, die sich über ein bis zu drei Zeichen erstrecken können, erhält man mit `\widetilde` und `\widehat`

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

Pfeile können ebenfalls über Zeichen oder ganze Formelausdrücke gesetzt werden:

`$\overleftarrow{A-B/C}$`	$\overleftarrow{A-B/C}$
`$\overrightarrow{A+B}$`	$\overrightarrow{A+B}$

Die normalen Akzentbefehle für den Textsatz `\",\~` ... können im mathematischen Satz nicht verwendet werden und umgekehrt.

5.7 Hervorgehobene Formeln — 'display-style'

Die bisher aufgetretenen Formeln können im fortlaufenden Text stehen, da sie noch nicht besonders hoch sind. Formeln, die mehr Platz benötigen, werden normalerweise in eine eigene Zeile gesetzt. Dazu wird die Formel bei der Eingabe in Doppel-Dollar (`$$`) eingeschlossen. Das Ergebnis wird dann zentriert gesetzt, mit Abstand zum oberen und unteren Text, aber ohne neuen Absatz im nachfolgenden Text. Für Formeln, die nicht zentriert, aber hervorgehoben werden sollen, gibt es ein Anwendungsbeispiel im Kapitel *Variationen des Formelsatzes*. Formeln im 'display-style' werden *nicht* getrennt. Hier muß der Autor bei der Eingabe selbst dafür Sorge tragen, daß die Formelteile in mehreren Zeilen angeordnet sind. Bei längeren Ausdrücken handelt es sich meist um Transformationen, die an mathematisch sinnvollen Positionen gruppiert werden sollten.

Die verwendeten Symbole in ihren Größenvarianten werden durch TEX automatisch bestimmt. Indizes und Exponenten werden ebenfalls an andere Positionen gesetzt. Ganz typisch ist der Unterschied bei der Behandlung von Integral- und Summenformeln.

Durch die Befehle `\textstyle` und `\displaystyle` kann unabhängig von `$` und `$$` der Übergang in ein anderes Layoutverhalten erzwungen werden.

Beispiel: Der Term $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ hat im *display-style* die Gestalt:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Die *text-style*-Eingabe hat die Form `$` ... `$`, die *display-style*-Eingabe `$$` ... `$$`. Es ist sofort zu erkennen, daß unterschiedliche Summensymbole verwendet werden und die Positionierung der oberen und unteren Grenzen anders erfolgt. Beim Satz des Bruches werden im 'textstyle' insgesamt kleinere Zeichen verwendet.

Die Eingabe für dieses Beispiel lautet übrigens:

```
$$\sum_{i=1}^n i= { n(n+1)\over 2}$$
```

5.8 Brüche

Es werden drei Arten von 'Brüchen' unterschieden:

— normale Brüche mit einem dünnen Bruchstrich, wie man sie gewohnt ist,
— 'Brüche' *ohne* Bruchstrich
— Brüche, deren Strichdicke explizit angegeben wird

Ein normaler Bruch wird mit dem Befehl `\over` eingegeben. Dabei wirkt `\over` logisch wie ein binärer Operator. Was links steht, *bis zur nächsten öffnenden Klammer* oder bis zu `$$`, kommt in den Zähler, was rechts steht *bis zur nächsten schließenden Klammer* oder `$$`, in den Nenner.

Soll ein Bruch von einem Manuskript erfaßt werden, so beginnt man zunächst mit einer Klammer '`{`', gibt die Befehle für den Zähler ein, dann `\over` für den Bruchstrich und anschließend die Befehle für den Nenner. Am Schluß wird mit einer schließenden Klammer '`}`' der Bruch beendet. Mehrere Brüche können durch 'Klammergebirge' ineinander verschachtelt werden.

Beispiele (im *display-style*)

```
$$ x+y^2 \over k + 1  $$
$$ a = x+y^{2 \over k+1} $$
$$ b = {x+y^2 \over k}+1 $$
$$ 1 \over { 1 + {1 \over x+1 }}$$
```

ergeben

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

$$a = x+y^{\frac{2}{k+1}}$$

$$b = \frac{x+y^2}{k}+1$$

$$\frac{1}{1+\frac{1}{x+1}}$$

Fügt man im letzten Beispiel noch einen `\displaystyle`-Befehl ein, so wird hierbei der Nenner nicht in Indexgröße, sondern in der Normalgröße gesetzt.

`$$ 1 \over { 1 +  \displaystyle{1 \over x+1 }}$$` ergibt dann:

$$\frac{1}{1+\frac{1}{x+1}}$$

Der normale Schrägstrich kann auch verwendet werden, um Doppelbrüche zu vermeiden. Die folgende Eingabe

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

ergibt

$$\frac{a/b}{\sqrt{1+c}}$$

In gleicher Weise wird `\atop` verwendet, nur entfällt hier bei der Ausgabe der Bruchstrich: `$$x \atop y+2 $$` führt zu

$$\begin{matrix} x \\ y+2 \end{matrix}$$

Und will man beim Bruchstrich auch noch angeben, wie dick er zu sein hat, benutzt man `\above`.
`$$ \displaystyle {a \over b} \above 1pt \displaystyle {c \over d} $$` ergibt das folgende Bild, wobei '1pt' dabei die Dicke des mittleren Bruchstrichs ist.

$$\begin{matrix} \frac{a}{b} \\ \rule{1pt}{0pt} \\ \frac{c}{d} \end{matrix}$$

Die `\displaystyle`-Befehle werden verwendet, um in den Teilbrüchen die große Form zu erhalten.

5.9 Binomial-Koeffizienten

Wie Brüche werden auch Binomial-Koeffizienten gesetzt: `\choose` heißt der passende Befehl. Praktisch sind dies Brüche mit Klammern ohne Bruchstriche.
`$$ n \choose k+1 $$` ergibt

$$\binom{n}{k+1}$$

Das textuelle Äquivalent nur mit eckigen beziehungsweise geschweiften Klammern ergibt sich mittels `$$ n \brack k+1 $$` zu

$$\begin{bmatrix} n \\ k+1 \end{bmatrix}$$

und `$$ n \brace k+1 $$` zu

$$\begin{Bmatrix} n \\ k+1 \end{Bmatrix}$$

5.10 Integralzeichen, Summen und andere Operatoren

Operatoren sind mathematische Funktionen, die auf eine Unterformel 'wirken'. In diesem Abschnitt sind 'große Operatoren' beschrieben, wie Summenzeichen und Integrale, die meist für sich allein stehen. Diese dominieren, allein durch ihre Größe, das Aussehen einer mathematischen Formel. Häufig besitzen sie untere und obere Grenzen, zum Beispiel in Summen und Integralen. Diese Grenzen werden mittels '`_`' für die untere Grenze und '`^`' für die obere Grenze angegeben. Es gilt auch hier der Grundsatz, daß zusammengehörende Teile mit '`{`' und '`}`' geklammert werden müssen.

Die folgende Tabelle enthält die standardmäßig vorhandenen Operatoren. Es besteht auch die Möglichkeit, eigene Operatoren zum Beispiel durch Zusammensetzen und Überlagern zu definieren (siehe dazu Abschnitt 9.2).

text-style	*display-style*	*Befehl*	*Verwendungsbeispiel*
$\sum_{\alpha}^{\omega}$	$\displaystyle\sum_{\alpha}^{\omega}$	`\sum`	Summenoperator
$\int_{\alpha}^{\omega}$	$\displaystyle\int_{\alpha}^{\omega}$	`\int`	Integral
$\int_{\alpha}^{\omega}$	$\displaystyle\intop_{\alpha}^{\omega}$	`\intop`	Integral
$\prod_{\alpha}^{\omega}$	$\displaystyle\prod_{\alpha}^{\omega}$	`\prod`	Produktzeichen
$\oint_{\alpha}^{\omega}$	$\displaystyle\oint_{\alpha}^{\omega}$	`\oint`	Ringintegral
$\coprod_{\alpha}^{\omega}$	$\displaystyle\coprod_{\alpha}^{\omega}$	`\coprod`	Koprodukt
$\bigcap_{\alpha}^{\omega}$	$\displaystyle\bigcap_{\alpha}^{\omega}$	`\bigcap`	Durchschnitt
$\bigcup_{\alpha}^{\omega}$	$\displaystyle\bigcup_{\alpha}^{\omega}$	`\bigcup`	Vereinigung
$\bigsqcup_{\alpha}^{\omega}$	$\displaystyle\bigsqcup_{\alpha}^{\omega}$	`\bigsqcup`	Supremum
$\bigvee_{\alpha}^{\omega}$	$\displaystyle\bigvee_{\alpha}^{\omega}$	`\bigvee`	Existenzquantor
$\bigwedge_{\alpha}^{\omega}$	$\displaystyle\bigwedge_{\alpha}^{\omega}$	`\bigwedge`	Allquantor
$\bigodot_{\alpha}^{\omega}$	$\displaystyle\bigodot_{\alpha}^{\omega}$	`\bigodot`	Produkt
$\bigotimes_{\alpha}^{\omega}$	$\displaystyle\bigotimes_{\alpha}^{\omega}$	`\bigotimes`	Tensorprodukt
$\bigoplus_{\alpha}^{\omega}$	$\displaystyle\bigoplus_{\alpha}^{\omega}$	`\bigoplus`	direkte Summe
$\biguplus_{\alpha}^{\omega}$	$\displaystyle\biguplus_{\alpha}^{\omega}$	`\biguplus`	disjunkte Vereinigung

Je nachdem, ob der Operator in einer Formel im Satz (*text-style*) oder als selbständige Formel (*display-style*) angewendet werden soll, erhält man ein unterschiedliches Aussehen. Die Nutzung eines Operators geschieht mittels des Steuersymbols für die Indexbildung '_' (Unterstrich), sowie mittels des Steuersymbols für die Exponentbildung '^' (Dach). Der Indexteil steht immer *unten*, der Exponentteil *oben* am gesetzten

Operatorsymbol. Je nach Stil und Formelgröße kann z.B. *unten* ein *"direkt unter"* oder auch ein *"rechts unten"* bedeuten.

In den folgenden Beispielen enthält die Eingabe nur den *text-style*, für den *display-style* müssen ja nur noch je ein Dollarzeichen davor und dahinter gesetzt werden.

`$ \sum_{n-1}^m $`	$\sum_{n-1}^{m}$	$\displaystyle\sum_{n-1}^{m}$
`$ \prod_{i=1}^n (i+1) $`	$\prod_{i=1}^{n}(i+1)$	$\displaystyle\prod_{i=1}^{n}(i+1)$
`$ \int_0^{2\pi} \sin x\, dx $`	$\int_0^{2\pi} \sin x\, dx$	$\displaystyle\int_0^{2\pi} \sin x\, dx$
`$ \prod_{j>0}\sum_{k>0}a_{jk} $`	$\prod_{j>0}\sum_{k>0} a_{jk}$	$\displaystyle\prod_{j>0}\sum_{k>0} a_{jk}$
`$ \bigcap_{i=1}^\infty M_i=\emptyset $`	$\bigcap_{i=1}^{\infty} M_i = \emptyset$	$\displaystyle\bigcap_{i=1}^{\infty} M_i = \emptyset$
`$\sum_{\scriptstyle i<m \atop j<m} x_{ij}$`	$\sum_{\substack{i<m \\ j<m}} x_{ij}$	$\displaystyle\sum_{\substack{i<m \\ j<m}} x_{ij}$

Bei der Verwendung dieser Operatoren ist es bei einigen von ihnen eine Stilfrage, wohin untere und obere Grenzen gesetzt werden. So wird beim Summenoperator `\sum` dies direkt unter und über das Summenzeichen geschehen, beim Integral ist es dagegen rechts unten und rechts oben neben dem Integralsymbol. Durch den Befehl `\limits` kann nun direkt angegeben werden, daß die Angaben für die Grenzen über und unter das Symbol gesetzt werden. Die Anweisung `\nolimits` bewirkt, daß sie neben das Operatorzeichen kommen. Von Bedeutung ist diese Angabe allerdings nur im 'displaystyle', da im 'textstyle' die Grenzen platzsparend neben das Zeichen gesetzt werden. Einige Beispiele von oben, wiederholt mit `\limits` und `\nolimits` Angaben, zeigen dann folgende Ausgaben:

`$$ \sum\nolimits_{n-1}^m $$`	$$\sum\nolimits_{n-1}^{m}$$
`$$ \prod\nolimits_{i=1}^n (i+1) $$`	$$\prod\nolimits_{i=1}^{n}(i+1)$$
`$$ \int\limits_0^\pi \sin x\,dx $$`	$$\int\limits_0^{\pi} \sin x\,dx$$

5.11 Klammern und Begrenzer

In TEX gibt es 29 verschiedene Sorten von Klammern und Begrenzern bzw. Trennzeichen. Bis auf wenige Ausnahmen können diese in beliebiger Größe gesetzt werden, da sie intern aus Teilzeichen zusammengesetzt sind. Bei der Verwendung gibt es zwei Möglichkeiten für die Größenbestimmung: die automatische Größenbestimmung durch "`\left`" und "`\right`" und die explizite Größenangabe.

Eingabe	*Ausgabe*	*Bemerkungen*
`(`	(	linke runde Klammer
`)`	)	rechte runde Klammer
`[` *oder* `\lbrack`	[	linke eckige Klammer
`]` *oder* `\rbrack`	]	rechte eckige Klammer
`\{` *oder* `\lbrace`	{	linke geschweifte Klammer
`\}` *oder* `\rbrace`	}	rechte geschweifte Klammer
`\lgroup`$_*$	(	runde Klammer Diese Klammer ist entstanden aus der geschweiften Klammer ohne deren Spitze.
`\rgroup`$_*$	)	runde Klammer, entstanden wie `\lgroup`
`\rmoustache`$_*$	⎱	‘Klammer’, entstanden aus geschweifter Klammer
`\lmoustache`$_*$	⎰	‘Klammer’, entstanden aus geschweifter Klammer
`\lfloor`	⌊	linke untere Ecke
`\rfloor`	⌋	rechte untere Ecke
`\lceil`	⌈	linke obere Ecke
`\rceil`	⌉	rechte obere Ecke
`\langle` *oder* `<`	⟨	linke spitze Klammer
`\rangle` *oder* `>`	⟩	rechte spitze Klammer
`/`	/	Schrägstrich
`\backslash`	\	inverser Schrägstrich
`|` *oder* `\vert`	|	vertikaler Strich
`\\|` *oder* `\Vert`	‖	doppelter vertikaler Strich (Norm)
`\arrowvert`$_*$	⏐	andere Form von ‘|’ entstanden aus ‘↓’
`\Arrowvert`$_*$	‖	andere Form von ‘‖’ entstanden aus ‘⇓’
`\bracevert`$_*$	⎪	senkrechter Strich Mittelstück einer Klammer
`\uparrow`	↑	Pfeil nach oben
`\downarrow`	↓	Pfeil nach unten
`\Uparrow`	⇑	Doppelpfeil nach oben
`\Downarrow`	⇓	Doppelpfeil nach unten
`\updownarrow`	↕	Pfeil nach oben und unten
`\Updownarrow`	⇕	Doppelpfeil nach oben und unten

Die mit '$_*$' markierten Befehle können nur in Kombination mit `\left` und `\right`, bzw. `\big...` verwendet werden, da diese Zeichen keine natürliche Anfangsgröße besitzen. Neben der Möglichkeit, TEX die Größe bestimmen zu lassen, kann die Klammergröße auch in fünf Abstufungen selbst gesetzt werden. Vor den Klammerbefehl wird ein Befehl zur Größenangabe gesetzt:

`\bigl`	`\bigr`	etwas größer als Normalgröße
`\Bigl`	`\Bigr`	1.5 mal so groß wie `\bigl` bzw. `\bigr`
`\biggl`	`\biggr`	2 mal so groß wie `\bigl` bzw. `\bigr`
`\Biggl`	`\Biggr`	2.5 mal so groß wie `\bigl` bzw. `\bigr`

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

liefert

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

Zum Größenvergleich die gebräuchlichsten Klammern. Folgende Eingaben

```
$$ \Biggl(\biggl(\Bigl(\bigl((0)\bigr)\Bigr)\biggr)\Biggr)$$
$$ \Biggl[\biggl[\Bigl[\bigl[[0]\bigr]\Bigr]\biggr]\Biggr]$$
$$ \Biggl\{\biggl\{\Bigl\{\bigl\{\{0
        \}\bigr\}\Bigr\}\biggr\}\Biggr\}$$
$$ \Biggl\lgroup\biggl\lgroup\Bigl\lgroup 0
              \Bigr\rgroup\biggr\rgroup\Biggr\rgroup$$
```

liefern:

$$\Biggl(\biggl(\Bigl(\bigl((0)\bigr)\Bigr)\biggr)\Biggr)$$

$$\Biggl[\biggl[\Bigl[\bigl[[0]\bigr]\Bigr]\biggr]\Biggr]$$

$$\Biggl\{\biggl\{\Bigl\{\bigl\{\{0\}\bigr\}\Bigr\}\biggr\}\Biggr\}$$

$$\Biggl\lgroup\biggl\lgroup\Bigl\lgroup 0\Bigr\rgroup\biggr\rgroup\Biggr\rgroup$$

Alle `\big...`-Befehle bestimmen gleichzeitig die satztechnische Funktion, also ob sie in einer Formel als linke oder rechte Klammer oder als Relation usw. gesetzt werden. Die Befehle `\bigl` ... beziehungsweise `\bigr` ... liefern jeweils linke und rechte Klammern. Für ein Zeichen mit der Eigenschaft einer Relation, heißen die entsprechenden Befehle `\bigm`, `\Bigm`, `\biggm` und `\Biggm` vorhanden. Das angehängte 'm' steht dabei für 'middle'.

Für *'normale Zeichen'* stehen `\big`, `\Big`, `\bigg` und `\Bigg` zur Verfügung.

5.12 Wachsende Klammern

TEX hat einen Mechanismus, um die erforderliche Größe eines Klammer-Paares zu bestimmen, das eine Formel umschließt. Man kann dieses Verfahren für jedes Delimiter-Paar nutzen, das man wünscht. Einzugeben ist lediglich:

`\left` <linke Klammer> <Formel> `\right` <rechte Klammer>

Durch die Angabe von `\left` und `\right` werden die Klammern in passender Größe bestimmt. Wenn `\left`,`\right` benutzt werden, müssen sie als Paar auftreten. Es dürfen aber zwei beliebige Klammern z.B. '[,)' benutzt werden. Soll eine der Klammern leer bleiben, so gibt es die Form '`\left.`' für eine leere linke Klammer und '`\right.`' für eine leere rechte Klammer. Stets müssen `\left` und `\right` als Paar auftreten. Es kommt zu einer Fehlermeldung, falls ein `\left` oder `\right` zu wenig oder zu viel verwendet wird.

Zu betonen ist, daß `\left` und `\right` nicht für ein 'Linkssymbol' wie ([{..., bzw. ein typisches "Rechtssymbol" wie)]}... stehen, sondern für *links von der Formel* und *rechts von der Formel*. Das den beiden Befehlen folgende Zeichen kann ein beliebiges Begrenzungssymbol sein.

Beispiele:

Eingabe	*textstyle*	*displaystyle*
`$ 1+\left( 1 \over 1-x^2 \right)^3 $`	$1+\left(\frac{1}{1-x^2}\right)^3$	$1+\left(\frac{1}{1-x^2}\right)^3$
`$ 1+\left\lgroup 1\over 1-x^2 \right\rgroup^3 $`	$1+\left(\frac{1}{1-x^2}\right)^3$	$1+\left(\frac{1}{1-x^2}\right)^3$
`$ 1+\left[ 1 \over 1-x^2 \right]^3 $`	$1+\left[\frac{1}{1-x^2}\right]^3$	$1+\left[\frac{1}{1-x^2}\right]^3$
`$ 1+\left\Vert f(x) \over1+g(x)\right\Vert$`	$1+\left\Vert\frac{f(x)}{1+g(x)}\right\Vert$	$1+\left\Vert\frac{f(x)}{1+g(x)}\right\Vert$
`$ 1+\left\{ f(x)\over1+g(x)\right\} $`	$1+\left\{\frac{f(x)}{1+g(x)}\right\}$	$1+\left\{\frac{f(x)}{1+g(x)}\right\}$

Bruchähnliche Strukturen, wie z.B. $\left\langle {n \atop k} \right\rangle$ können mit beliebigen Klammern definiert werden: Es gibt drei Basisbefehle für 'geklammerte Brüche'

`\overwithdelims` <öffnende Klammer> <schließende Klammer>

`\atopwithdelims` <öffnende Klammer> <schließende Klammer>

`\abovewithdelims` <öffnende Klammer> <schließende Klammer> <Dimension>

Dabei ist `\overwithdelims` der Befehl für eine Ausgabe mit Bruchstrich. Eine Formel ohne Bruchstrich wird durch `\atopwithdelims` erreicht. Mittels `\abovewithdelims` wird ein Bruch gesetzt, dessen Bruchstrichstärke als dritter Parameter angegeben wird. Der letzte Befehl entspricht dem Befehl `\above`, der im Abschnitt 'Brüche' beschrieben ist.

Anwendungsbeispiele:

`$ n\atopwithdelims\langle\rangle k$`	$\left\langle {n \atop k} \right\rangle$
`$ n\overwithdelims\langle\rangle k$`	$\left\langle \frac{n}{k} \right\rangle$
`$ n\abovewithdelims\langle\rangle 1pt k$`	$\left\langle \frac{n}{k} \right\rangle$

5.13 Mathematische Funktionen

Es gibt eine Reihe mathematische Funktionen, die in Veröffentlichungen üblicherweise nicht in der *italic* Schrift gesetzt werden, wie etwa 'sin α' oder 'cos β'. Wenn diese Funktionen 'untere Grenzen' besitzen, werden diese wie sonst üblich mittels '_' eingegeben. Dabei besitzen sie von vornherein eine sinnvolle Plazierung für die unteren Grenzen:

`$$\max_{1<i<n} \log_2 a_i$$` zeigt die Unterschiede

$$\max_{1<i<n} \log_2 a_i$$

Die folgende Liste führt die vorhandenen mathematischen Funktionen auf. Einträge, die mit einem '$_*$' markiert sind, stehen für Funktionen, die ihre Delimiter direkt unter ihrem Namen erhalten:

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

Anwendungsergebnisse:

`$\sin2\beta=2\sin\beta\cos\beta$`	$\sin 2\beta = 2\sin\beta\cos\beta$
`$O(n\log n\log\log n)$`	$O(n\log n\log\log n)$
`$\max_{1\le n}\log_2P_n$`	$\max_{1\le n}\log_2 P_n$
`$$\min_{1\le n}\log_2P_n$$`	$\min\limits_{1\le n}\log_2 P_n$
`$$\liminf_{n\to\infty}x_n=0$$`	$\liminf\limits_{n\to\infty} x_n = 0$

Möchte man sich eigene Befehle mit entsprechendem Verhalten definieren, so sei hier zunächst das Rezept dazu angegeben und auf die Erläuterungen in späteren Abschnitten verwiesen: `\min` ist definiert durch `\def\min{\mathop{\rm min}}` und `\log` ist definiert durch `\def\log{\mathop{\rm log}\nolimits}`.

Noch ein Hinweis: Alle diese mathematischen Funktionen werden beim Satz als *(große) Operatoren* aufgefaßt und entsprechend behandelt.

5.14 Abstände

Ist man mit den von TEX bestimmten Abständen zwischen einzelnen Formelelementen nicht zufrieden, bieten folgende Befehle Änderungsmöglichkeiten:

`\,`	um 3/18 "quad" größerer Abstand ⇔ `\thinmuskip`
`\>`	um 4/18 "quad" größerer Abstand ⇔ `\medmuskip`
`\;`	um 5/18 "quad" größerer Abstand ⇔ `\thickmuskip`
`\!`	um 3/18 "quad" kleinerer Abstand ⇔ `-\thinmuskip`

Eingabe	*textstyle*	*displaystyle*
`$ \int\!\!\!\int_D f(x,y)\,dx\,dy $`	$\iint_D f(x,y)\,dx\,dy$	$\displaystyle\iint_D f(x,y)\,dx\,dy$

gegenüber der ungeänderten Fassung:

`$ \int\int_D f(x,y) dx  dy $`	$\int\int_D f(x,y)dxdy$	$\displaystyle\int\int_D f(x,y)dxdy$

Es gibt eine Reihe von Beeinflussungsmöglichkeiten für die Abstände beim Mathematiksatz. Abstände können zum Beispiel mit `\quad` oder `\qquad` in einer Formel eingefügt werden. Häufig geschieht dies in Kombination mit Text:

```
$ f(x)=\sin x \quad \hbox{f\"ur} \quad x>0$
```

liefert

$f(x) = \sin x \quad \text{für} \quad x > 0$

Hier ist noch eine Anmerkung zum Text innerhalb mathematischer Formeln zu machen. Am saubersten ist immer der Einschluß des Textes in eine `\hbox`, da in dieser `\hbox` das Satzverhalten für Textsatz gilt. Sonst hat man meist einige Dinge zu beachten: Die Abstände zwischen den Wörtern werden nach Mathematikkonventionen bestimmt. Die gewählte Schrift ist eine spezielle Mathematikschrift für Symbole; diese enthält zum Beispiel keine Ligaturen für etwa 'ff'. Umlaute, die zum Beispiel durch '`\"a`' eingegeben werden, sind im Mathematikmodus nicht erlaubt, nur "mathematische Akzente" (siehe 5.6) sind möglich.

5.15 Matrizen

Matrizen werden von Mathematikern in vielfältiger Form genutzt. Im allgemeinen handelt es sich dabei um eine rechteckige Anordnung von Termen — eine *Matrix*. Eine Standardmatrix ist zum Beispiel folgender Ausdruck:

$$A = \left(\begin{array}{ccc} x-\lambda & 1 & 0 \\ 0 & x-\lambda & 1 \\ 0 & 0 & x-\lambda \end{array} \right)$$

Dabei erfolgte die Eingabe als:

```
$$ A = \left(  \matrix{ x-\lambda & 1          & 0          \cr
                        0         & x-\lambda & 1          \cr
                        0         & 0         & x-\lambda \cr}
       \right) $$
```

Die einzelnen Matrixzeilen werden durch den Befehl '`\cr`' getrennt. Die einzelnen Matrixelemente durch das Zeichen '`&`'. Die Größe der Matrix wird dann automatisch aufgrund der größten Elemente bestimmt. Die linken und rechten Klammern wurden durch '`\left(`' und '`\right)`' erwirkt. (Alle Leerzeichen zwischen den `$$`-Symbolen werden von TEX überlesen. Das gleiche gilt für die Anordnung in Zeilen.)

Da die obige Matrix eine Standardform ist, gibt es für sie eine abgekürzte Schreibweise `\pmatrix` (*parenthesized matrix*).

```
$$ A = \pmatrix{ x-\lambda & 1          & 0          \cr
                 0         & x-\lambda & 1          \cr
                 0         & 0         & x-\lambda \cr}$$
```

Matrizen können mit verschiedenen Klammern gesetzt werden. Lediglich durch '`\left`' und '`\right`' werden die Klammern vorgegeben. Die Liste der möglichen Klammern finden Sie im Abschnitt 'Klammern und Begrenzer'. Bei der Eingabe ist zu beachten,

daß die Reihenfolge `\left` und `\right` nicht vertauscht wird. `\left` und `\right` müssen auch immer als Paar auftreten, da sonst die Größe der inneren Formel nicht berechnet werden kann. Möchte man eine nichtpaarige Struktur, so gibt es die *leere Klammer* ('.') – dargestellt durch einen Punkt. '`\right.`' z.B. erzeugt eine leere rechte Klammer. Noch ein Beispiel: Die Determinante

$$\det A = \left| \begin{matrix} x-\lambda & 1 & 0 \\ 0 & x-\lambda & 1 \\ 0 & 0 & x-\lambda \end{matrix} \right|$$

wurde durch

```
$$ \det  A =
   \left\vert \matrix{ x-\lambda  & 1          & 0          \cr
                       0          & x-\lambda  & 1          \cr
                       0          & 0          & x-\lambda  \cr}
   \right\vert$$
```

erzeugt.

Wichtig sind noch 'Pünktchen' in einer Matrix, die durch die Befehle

`\ldots`	$\ldots$	*low dots*
`\vdots`	$\vdots$	*vertical dots*
`\ddots`	$\ddots$	*diagonal dots*
`\cdots`	$\cdots$	*centered dots*

erzeugt werden. Illustriert sei dies durch

```
$$A=\pmatrix{ a_{11} & a_{12} & \ldots & a_{1n}  \cr
              a_{21} & a_{22} & \ldots & a_{2n}  \cr
              \vdots & \vdots & \ddots & \vdots  \cr
              a_{n1} & a_{n2} & \ldots & a_{nn}  \cr}$$
```

Dies ergibt

$$A = \pmatrix{ a_{11} & a_{12} & \ldots & a_{1n} \cr a_{21} & a_{22} & \ldots & a_{2n} \cr \vdots & \vdots & \ddots & \vdots \cr a_{n1} & a_{n2} & \ldots & a_{nn} \cr}$$

Bisweilen werden Matrizen noch durch waagerechte und senkrechte Striche aufgeteilt: Hierzu werden `\hrule` und `\vrule` benötigt. Für die senkrechten Striche wird eine eigene Matrixspalte definiert. Der waagerechte Strich wird mit `\noalign{\hrule}` gezogen. Das Problem bei solchen Strukturen ist die notwendige Höhe der senkrechten Linien.

Die Eingabe

```
$$  A_3 = \left[
    \matrix{ & &   & & \vrule & a_{11} & \ldots & a_{1n}  \cr
             & & C & & \vrule & a_{1n} & \ldots & a_{nn}  \cr
             & &   & & \vrule & a_{1n} & \ldots & a_{nn}  \cr
           \noalign{\hrule}
             & &   & & \vrule & \lambda&        &         \cr
             & & I & & \vrule &        & \ddots &         \cr
             & &   & & \vrule &        &        & \lambda \cr
           } \right] $$
```

ergibt

$$A_3 = \left[\begin{array}{ccc|ccc} & & & a_{11} & \ldots & a_{1n} \\ & C & & a_{1n} & \ldots & a_{nn} \\ & & & a_{1n} & \ldots & a_{nn} \\ \hline & & & \lambda & & \\ & I & & & \ddots & \\ & & & & & \lambda \end{array}\right]$$

Allerdings sind die vertikalen Linien durch Leerräume unterbrochen. Dies liegt daran, daß die Zeilen auch hier abhängig vom Parameter '`\baselineskip`' gesetzt werden. (Dieses Problem wird ausführlich im Kapitel 'Tabellensatz' diskutiert.)

Ein Rezept, um die Löcher zu umgehen, ist folgender Befehl `\vrule height ... pt depth ... pt`, sowie ein Zurückskippen mittels `\noalign{\vskip - ...pt}`. Die nötigen Werte für die Höhenangaben sind von Fall zu Fall zu wählen. Günstig ist eine Überlappung der Linien. (`\noalign` ist in Kapitel 6 noch näher erläutert.)

Die Variation der Eingabe zu

```
$$  \def\linie{\vrule height 17pt depth 5pt}
    \def\back{\noalign{\vskip-3pt}}
    \lineskip=0pt
 A_3 = \left[
       \matrix{ & &   & & \linie & a_{11} & \ldots & a_{1n}  \cr
              \back
                & & C & & \linie & a_{1n} & \ldots & a_{nn}  \cr
              \back
                & &   & & \linie & a_{1n} & \ldots & a_{nn}  \cr
              \noalign{\hrule}
                & &   & & \linie & \lambda&        &         \cr
              \back
                & & I & & \linie &        & \ddots &         \cr
              \back
                & &   & & \linie &        &        & \lambda \cr}
       \right] $$
```

liefert

$$A_3 = \left[\begin{array}{c|ccc} & a_{11} & \dots & a_{1n} \\ C & a_{1n} & \dots & a_{nn} \\ & a_{1n} & \dots & a_{nn} \\ \hline & \lambda & & \\ I & & \ddots & \\ & & & \lambda \end{array}\right]$$

Zu bemerken sind noch die Abkürzungen

```
\def\linie{\vrule height 14pt depth 5pt}
\def\back{\noalign{\vskip-3pt}}
```

Damit kann mittels der Befehle `\linie` und `\back` jeweils der komplette Text als Makroaufruf referiert werden.

Normalerweise sind Matrizen so groß, daß sie nur im *displaystyle*, d.h. für sich auftreten. Gelegentlich möchte man *sehr kleine* Matrizen auch im Text setzen, zum Beispiel $\binom{1\,1}{0\,1}$. Die vorangehende Matrix wurde durch Anwendung der Binomialoperation `\choose` im Befehl `$1\,1\choose0\,1$` erzeugt. Die Befehle `\,` sorgen nur für den größeren Abstand der Elemente. Eine Alternative ist auch mittels `\atop` – *Bruch ohne Bruchstrich* – mehrere Elemente hintereinander anzuordnen: $\binom{a\ b\ c}{l\ m\ n}$. Die Eingabe dazu lautet

```
$\bigl( { a\atop l} {b \atop m} {c \atop n} \bigr)$
```

Dabei erzeugen `\bigl(` und `\bigr)` etwas größere Klammern.

5.16 Einzelne gruppierende Klammern

In Definitionen sind Gruppierungen verschiedener Fallgruppen beliebt. Diese haben eine 'matrixähnliche' Struktur mit 'fehlender' rechter Klammer:

$$|x| = \begin{cases} x & \text{falls } x \ge 0 \\ -x & \text{sonst} \end{cases}$$

wird durch

```
$$\vert x\vert = \cases{x   &falls $x\ge0$ \cr
                        -x  & sonst\cr }$$
```

gesetzt. Dabei ist `\cases{ ... }` der Befehl, der die passend große öffnende Klammer mit der folgenden Information erzeugt. Jede 'Fallzeile' wird durch `\cr` getrennt, ein Tabulatorsymbol '`&`' trennt jeweils die einzelnen Spalten. '`\cases`' ist dabei eine Matrix, die als Klammern '`\left\{`' und '`\right.`' besitzt. Die erste Matrixspalte wird dabei automatisch im Mathematikmodus und die zweite im normalen Textmodus gesetzt.

Die andere Form einer gruppierenden Klammer sind die horizontal liegenden Klammern, die mit `\overbrace` und `\underbrace` erzeugt werden.

```
$$\overbrace{x+\cdots+x}^{k\rm\;mal}$$
$$\underbrace{x+y+z}_{>0}$$
```

erzeugt

$$\overbrace{x+\cdots+x}^{k\ \mathrm{mal}}$$

$$\underbrace{x+y+z}_{>0}$$

Noch ein Hinweis: Gleich aussehende Klammern, die unter `\upbracefill` und `\downbracefill` referiert werden können, sind noch für den normalen Textsatz vorhanden. Diese werden automatisch so breit, wie es die Umgebung etwa in einer Tabelle erfordert. Näheres ist dazu im Abschnitt 6.7 unter Tabellensatz zu finden.

5.17 Mehrzeilige Formeln — Ausrichtung

Oft besteht der Wunsch, mehrere Gleichungen, die in aufeinanderfolgenden Zeilen gesetzt werden, nach einem bestimmten Element auszurichten. Meistens ist dies das Gleichheitszeichen.

$$X_1 + \cdots + X_p = m$$
$$Y_1 + \cdots + Y_q = m$$

Dafür gibt es den Befehl `\eqalign`. Die einzelnen Zeilen werden durch '`\cr`' wie beim Tabellensatz getrennt. Vor das Zeichen, nach dem die linken und rechten Seiten ausgerichtet werden sollen, wird ein '`&`' gesetzt. Die Eingabe für das vorangehende Beispiel lautet:

```
$$\eqalign{X_1 + \cdots + X_p &=m \cr
           Y_1 + \cdots + Y_q &=m \cr}$$
```

Sollen mehrere Formelblöcke gegeneinander ausgerichtet werden, so kann dies durch mehrfache Anwendung von `\eqalign` im gleichen `$$`-Block geschehen.

```
$$\eqalign{X_1 + \cdots + X_p &=m \cr
           Y_1 + \cdots + Y_q &=m \cr}
   \qquad                                % Abstand
   \eqalign{\alpha &= f(z) \cr
            \beta  &= f(z^2) \cr
            \gamma &= f(z^3) \cr}
$$
```

ergibt

$$\eqalign{X_1 + \cdots + X_p &=m \cr
           Y_1 + \cdots + Y_q &=m \cr}
   \qquad
   \eqalign{\alpha &= f(z) \cr
            \beta &= f(z^2) \cr
            \gamma &= f(z^3) \cr}$$

Das `\qquad` zwischen den beiden `\eqalign`-Aufrufen bestimmt den Abstand zwischen den beiden Formelblöcken.

Sollen zusätzlich um die einzelnen Formelblöcke Klammern stehen, so ist `\left` und `\right` zu setzen. Die mit `\left\{` und `\right\}` abgewandelte Eingabe

```
$$\left\{\eqalign{X_1 + \cdots + X_p &=m \cr
                  Y_1 + \cdots + Y_q &=m \cr}\right\}
        \qquad                                % Abstand
  \left\{\eqalign{\alpha &= f(z) \cr
                  \beta  &= f(z^2) \cr
                      \gamma &= f(z^3) \cr}\right\}
$$
```

liefert

$$\left\{\begin{aligned} X_1 + \cdots + X_p &= m \\ Y_1 + \cdots + Y_q &= m \end{aligned}\right\} \qquad \left\{\begin{aligned} \alpha &= f(z) \\ \beta &= f(z^2) \\ \gamma &= f(z^3) \end{aligned}\right\}$$

5.18 Numerierte Formeln

In mathematischen Arbeiten ist es üblich, Formeln zu numerieren, damit diese besser referiert werden können. TEX bietet die Möglichkeit, Formeln sowohl auf der linken als auch auf der rechten Seite zu numerieren. Die einfachste Form, eine *einzelne* Formel zu numerieren, ist `\eqno` für rechtsseitige Numerierung und `\leqno` für linksseitige Numerierung:

```
$$<Formel>\eqno<Numerierungsformel>$$
$$<Formel>\leqno<Numerierungsformel>$$
```

```
$$\sin 18^\circ= {1\over4} (\sqrt5-1)\eqno(1)$$
$$\sum_{i=1}^n i = {n(n+1) \over 2} \leqno(2')$$
```

liefert

$$\sin 18^\circ = \frac{1}{4}(\sqrt{5}-1) \tag{1}$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{2'}$$

Um mehrzeilige, ausgerichtete Gleichungen zu numerieren, gibt es Abwandlungen des Befehls `\eqalign`. Dies sind `\eqalignno` mit einer Numerierung auf der rechten Seite und `\leqalignno` mit einer Numerierung auf der linken Seite. Die Eingabeform ist wie bei `\eqalign` um ein weiteres `&`-Zeichen je Formelzeile (bis `\cr`) erweitert. Hinter dem zweiten Tab-Zeichen je Formelzeile folgt der Teil für die Numerierung.
Die Eingabe

```
$$\eqalignno{ \sin^2 \alpha + \cos^2 \alpha &= 1 &(1) \cr
                   a^2 + b^2 & = c^2             &(*) \cr}
$$
```

erzeugt

$$\sin^2\alpha + \cos^2\alpha = 1 \tag{1}$$
$$a^2 + b^2 = c^2 \tag{$*$}$$

Es sind zwei Tab-Zeichen (‘&’) je Zeile vorhanden. Der Text nach dem zweiten Zeichen wird für die Numerierung verwendet. `\leqalignno` hat die gleiche Aufrufform, die im zweiten Teil angegebene Numerierung erscheint lediglich links.

```
$$\leqalignno{ \sin^2 \alpha + \cos^2 \alpha &= 1 &(2) \cr
                    a^2 + b^2 & = c^2              &(**) \cr}
$$
```

liefert

$$\sin^2\alpha + \cos^2\alpha = 1 \tag{2}$$
$$a^2 + b^2 = c^2 \tag{$**$}$$

Es ist nicht notwendig, alle Formeln zu numerieren.

```
$$\eqalignno{ (x+y)(x-y) &= x^2-xy+yx-y^2 & \cr   %ohne Nummer
                         &= x^2-y^2       &(1) \cr
                (x+y)^2  &= x^2+2xy+y^2   &(2) \cr} $$
```

liefert

$$\begin{aligned}(x+y)(x-y) &= x^2 - xy + yx - y^2 \\ &= x^2 - y^2 \qquad (1)\\ (x+y)^2 &= x^2 + 2xy + y^2 \qquad (2)\end{aligned}$$

Die Nummer der Formel wird selbst im Mathematikmodus gesetzt, das heißt es gelten die Umbruchregeln des Mathematikteiles. Möchte man jedoch in die Numerierung ‘richtigen’ Text setzen, muß eine `\hbox` um sie gesetzt werden.

```
$$\int_{-\infty}^\infty e^{-x^2}\,dx = \pi
                                  \eqno \hbox{(Formel A)}$$
```

liefert

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \pi \qquad \text{(Formel A)}$$

Neben den bisher erwähnten mathematischen Zeichen gibt es eine Reihe von Symbolen, die zur Referenz ohne nähere Erläuterungen aufgeführt werden sollen:

5.19 Mathematische binäre Operatoren

Beim Setzen mathematischer binärer Operatoren wird rechts und links des Operators gleich viel Platz gelassen. Neben den Grundoperatoren $+ - *$ gibt es noch eine Reihe weiterer Operatoren:

$\pm$	`\pm`	$\cap$	`\cap`	$\vee$	`\vee` *oder* `\lor`
$\mp$	`\mp`	$\cup$	`\cup`	$\wedge$	`\wedge` *oder* `\land`
$\setminus$	`\setminus`	$\uplus$	`\uplus`	$\oplus$	`\oplus`
$\cdot$	`\cdot`	$\sqcap$	`\sqcap`	$\ominus$	`\ominus`
$\times$	`\times`	$\sqcup$	`\sqcup`	$\otimes$	`\otimes`
$*$	`\ast`	$\triangleleft$	`\triangleleft`	$\oslash$	`\oslash`
$\star$	`\star`	$\triangleright$	`\triangleright`	$\odot$	`\odot`
$\diamond$	`\diamond`	$\wr$	`\wr`	$\dagger$	`\dagger`
$\circ$	`\circ`	$\bigcirc$	`\bigcirc`	$\ddagger$	`\ddagger`
$\bullet$	`\bullet`	$\bigtriangleup$	`\bigtriangleup`	$\amalg$	`\amalg`
$\div$	`\div`	$\bigtriangledown$	`\bigtriangledown`		

5.20 Mathematische Relationen

Neben den Standardrelationen $< >$ und $=$ gibt es noch

$\leq$	`\leq` *oder* `\le`	$\geq$	`\geq` *oder* `\ge`	$\equiv$	`\equiv`
$\prec$	`\prec`	$\succ$	`\succ`	$\sim$	`\sim`
$\preceq$	`\preceq`	$\succeq$	`\succeq`	$\simeq$	`\simeq`
$\ll$	`\ll`	$\gg$	`\gg`	$\asymp$	`\asymp`
$\subset$	`\subset`	$\supset$	`\supset`	$\approx$	`\approx`
$\subseteq$	`\subseteq`	$\supseteq$	`\supseteq`	$\cong$	`\cong`
$\sqsubseteq$	`\sqsubseteq`	$\sqsupseteq$	`\sqsupseteq`	$\bowtie$	`\bowtie`
$\in$	`\in`	$\ni$	`\ni` *oder* `\owns`	$\propto$	`\propto`
$\vdash$	`\vdash`	$\dashv$	`\dashv`	$\models$	`\models`
$\smile$	`\smile`	$\mid$	`\mid`	$\doteq$	`\doteq`
$\frown$	`\frown`	$\parallel$	`\parallel`	$\perp$	`\perp`

Eine Reihe von Relationen sind *verneint* auch sinnvoll:

$\not<$	`\not<`	$\not>$	`\not>`	$\neq$	`\ne`, `\not=` *oder* `\neq`
$\not\leq$	`\not\leq`	$\not\geq$	`\not\geq`	$\not\equiv$	`\not\equiv`
$\not\prec$	`\not\prec`	$\not\succ$	`\not\succ`	$\not\sim$	`\not\sim`
$\not\preceq$	`\not\preceq`	$\not\succeq$	`\not\succeq`	$\not\simeq$	`\not\simeq`
$\not\subset$	`\not\subset`	$\not\supset$	`\not\supset`	$\not\approx$	`\not\approx`
$\not\subseteq$	`\not\subseteq`	$\not\supseteq$	`\not\supseteq`	$\not\cong$	`\not\cong`
$\not\sqsubseteq$	`\not\sqsubseteq`	$\not\sqsupseteq$	`\not\sqsupseteq`	$\not\asymp$	`\not\asymp`
$\notin$	`\not\in`	$\not\ni$	`\not\ni` *oder* `\not\owns`	$\not\propto$	`\not\propto`
$\not\vdash$	`\not\vdash`	$\not\dashv$	`\not\dashv`	$\not\models$	`\not\models`
$\not\smile$	`\not\smile`	$\not\mid$	`\not\mid`	$\not\doteq$	`\not\doteq`
$\not\frown$	`\not\frown`	$\not\parallel$	`\not\parallel`	$\not\perp$	`\not\perp`

Das durch `\not\in` gebildete Zeichen "$\not\in$" hat den Schrägstrich etwas weit rechts. Durch den Sonderbefehl `\notin` kann eine verbesserte Fassung "$\notin$" aufgerufen werden. Pfeile werden wie andere Relationen behandelt.

$\leftarrow$	`\leftarrow`, `\gets`	$\longleftarrow$	`\longleftarrow`	$\uparrow$	`\uparrow`
$\Leftarrow$	`\Leftarrow`	$\Longleftarrow$	`\Longleftarrow`	$\Uparrow$	`\Uparrow`
$\rightarrow$	`\rightarrow`, `\to`	$\longrightarrow$	`\longrightarrow`	$\downarrow$	`\downarrow`
$\Rightarrow$	`\Rightarrow`	$\Longrightarrow$	`\Longrightarrow`	$\Downarrow$	`\Downarrow`
$\leftrightarrow$	`\leftrightarrow`	$\longleftrightarrow$	`\longleftrightarrow`	$\updownarrow$	`\updownarrow`
$\Leftrightarrow$	`\Leftrightarrow`	$\Longleftrightarrow$	`\Longleftrightarrow`	$\Updownarrow$	`\Updownarrow`
		$\iff$	`\iff` *mit extra Leerplatz !*		
$\mapsto$	`\mapsto`	$\longmapsto$	`\longmapsto`	$\nearrow$	`\nearrow`
$\hookleftarrow$	`\hookleftarrow`	$\hookrightarrow$	`\hookrightarrow`	$\searrow$	`\searrow`
$\leftharpoonup$	`\leftharpoonup`	$\rightharpoonup$	`\rightharpoonup`	$\swarrow$	`\swarrow`
$\leftharpoondown$	`\leftharpoondown`	$\rightharpoondown$	`\rightharpoondown`	$\nwarrow$	`\nwarrow`
$\rightleftharpoons$	`\rightleftharpoons`				

Um etwas Ordnung in die Pfeilbefehle zu bringen: Ein Großbuchstabe am Anfang deutet Doppelpfeile an. Ein vorgesetztes '`long`' die lange Variante. Es gibt Pfeile *(arrow)*, Haken *(hook)* und 'Harpunen' *(harpoon)*. Die Richtungen werden mit `up`, `down`, `left` und `right` sowie `ne` *(north east)*, `se` *(south east)*, `sw` *(south west)* und `nw` *(north west)* verdeutlicht. Senkrechte Striche (`up` und `down`) können als Klammern beliebig groß werden. Sie werden dann in Kombination mit `\left`, `\right` oder `\big..` gesetzt.

Um über Relationen Texte zu setzen, gibt es den Befehl `\buildrel`. Dabei liegt folgende Syntax vor:

`\buildrel` *oberer Text* `\over` *Relation*

Beispielsweise liefern

```
$$ \buildrel \alpha\beta \over \longrightarrow   $$
$$ \buildrel \rm def \over = $$
```

$$\buildrel \alpha\beta \over \longrightarrow$$
$$\buildrel \rm def \over =$$

5.21 Sonstige mathematische Symbole

$\aleph$	`\aleph`	$\prime$	`\prime`	$\forall$	`\forall`
$\hbar$	`\hbar`	$\emptyset$	`\emptyset`	$\exists$	`\exists`
$\imath$	`\imath`	∇	`\nabla`	$\neg$	`\neg`
$\jmath$	`\jmath`	$\surd$	`\surd`	$\flat$	`\flat`
ℓ	`\ell`	$\top$	`\top`	$\natural$	`\natural`
$\wp$	`\wp`	$\bot$	`\bot`	$\sharp$	`\sharp`
$\Re$	`\Re`	$\Vert$	`\Vert`	$\clubsuit$	`\clubsuit`
$\Im$	`\Im`	$\angle$	`\angle`	$\diamondsuit$	`\diamondsuit`
∂	`\partial`	$\triangle$	`\triangle`	$\heartsuit$	`\heartsuit`
∞	`\infty`	$\backslash$	`\backslash`	$\spadesuit$	`\spadesuit`
$\S$	`\S`	$\P$	`\P`	$\dagger$	`\dag`
$\ddagger$	`\ddag`	$\vert$	`\vert`		

Kalligraphische große Buchstaben $\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$ erhält man mit `${\cal A}$ ... ${\cal Z}$`.

'Oldstyle'-Ziffern 0123456789 erhält man mittels `{\oldstyle 0123456789}`.

5.22 Leeroperatoren

Gelegentlich hat der Autor ganz besondere Vorstellungen von einem mathematischen Operationszeichen, das auch durch Übereinanderlegen anderer Zeichen nicht gebildet werden kann. Man möchte dann *freien* Platz lassen, um später evtl. von Hand das private 'Superzeichen' einzusetzen. Diese Leistung bietet der Befehl `\phantom`. Die Eingabe `\phantom{08}15` bewirkt, daß genauso gesetzt wird, als ob `0815` eingegeben wurde. Lediglich bleibt das "`08`" unsichtbar.

Für Operatoren ist der Sachverhalt ein wenig komplizierter: Neben dem Platz, den diese benötigen, wird bei ihnen auch noch gesteuert, wohin Index- und Exponentteile gesetzt werden. Bei `\sum` und `\int` sind dies schon unterschiedliche Stellen. Durch

```
\mathop{\phantom\sum}
```

wird wird dies erreicht. Das Summensymbol `\sum` wird *nicht* gesetzt, aber das Verhalten bleibt erhalten.
Beispielsweise

```
$$\mathop{\phantom\sum}_{i=1}^\infty x_n = \pi$$
$$\mathop{\phantom\int}_{-\infty}^\infty e^{-x^2} = \pi$$
```

liefert

$$\mathop{\phantom\sum}_{i=1}^\infty x_n = \pi$$

$$\mathop{\phantom\int}_{-\infty}^\infty e^{-x^2} = \pi$$

Neben `\mathop`, mit dem ein Verhalten als *(großer) mathematischer Operator* definiert wird, gibt es noch weitere Befehle zur Funktionsfestlegung. Diese seien hier noch einmal kurz aufgeführt (Näheres im Abschnitt 9.1 *"Abstandssteuerung"*).

`\mathord`	normales Zeichen	`\mathopen`	linke Klammer
`\mathop`	Operator	`\mathclose`	rechte Klammer
`\mathbin`	binäre Relation	`\mathpunct`	Satzzeichen
`\mathrel`	Relation	`\mathinner`	Unterformel

Der verwendete Befehl `\phantom` hat die Wirkung, Platz exakt in Höhe und Breite und Tiefe zu lassen, als wenn dieses Zeichen selbst dort stände. Neben `\phantom` gibt

es noch weitere Befehle, um ähnliche Effekte zu erreichen: `\hphantom`,`\vphantom` und `\smash`. Die folgende Tabelle stellt die Anweisungen zusammen:

Befehl	*Höhe und Tiefe*	*Breite*	*Ausgabe*
`\phantom {... text ...}`	wie Original	wie Original	Leerraum
`\hphantom {... text ...}`	0	wie Original	Leerraum
`\vphantom {... text ...}`	wie Original	0	Leerraum
`\smash {... text ...}`	0	wie Original	**wie Original**

Die als Parameter angebene Information zu `\phantom` usw. kann auch normaler Text sein. Die Befehle überprüfen dies automatisch und verbleiben entweder im Text- oder im Mathematikmodus.

5.23 Lange Formeln — Formeltrennung

Zunächst ist zu sagen, daß der Autor eigentlich derjenige ist, der weiß, wo seine Formeln getrennt werden können. Die beste Lösung ist immer die explizite Trennvorgabe durch ihn; denn er hat das Verständnis bezüglich Funktion und Symmetrien seiner Formelbestandteile.

Einige Dinge werden jedoch durch TEX auch automatisch, beziehungsweise halbautomatisch bei *Formeln im Text* vollzogen: Automatisch werden Formeln *im text style* nach einer mathematischen Relation "$< \ = \ > \ldots$" oder einem binären Operator "$+ - * \ldots$" getrennt. Dies geschieht aber nur, wenn diese Operatoren jeweils auf dem äußersten Niveau einer Klammerverschachtelung stehen.
Bei der Eingabe

```
$g(x,y)=x^2+2xy+y^2=(x+y)(x+y)$
```

kann nach den Gleichheitszeichen und Pluszeichen getrennt werden. Dagegen wird bei

```
$g(x,y)={x^2+2xy+y^2}={(x+y)(x+y)}$
```

nur noch an den Gleichheitszeichen getrennt. Man beachte auch den Unterschied zwischen den beiden Eingaben

```
$   f(x) = x-2   $
$ { f(x) = x-2 } $
```

Die zweite Formel kann nicht getrennt werden.

Zur Trennunterstützung hat der Autor die Möglichkeit, Vortrenner einzugeben. Der Befehl dazu heißt `\allowbreak`. Auch dieser Befehl hat nur im äußersten Klammerniveau Gültigkeit. Wie im normalen Textsatz verbietet der Befehl `\nobreak` die Trennung an bestimmten Positionen, was unter Umständen nach Gleichheitszeichen sinnvoll sein kann. Für multiplikative Terme gibt es noch eine besondere Form des Vortrenners "`\*`", der wie ein "`\-`" im Text wirkt. Als "Trennsymbol" wird das $\times$-Zeichen eingesetzt (`\times`). Eine Anwendung ist

```
$(x_1+1) \* (x_2+2) \* (x_3+3) ... $
```

Hervorgehobene Formeln im *display-style* werden *nicht* automatisch getrennt. Die stilistisch angemessene Form der Trennung ist *vor* einem binären Operator mit einer Einrückung des zweiten Formelteils (`\qquad`).

Beispiel:

```
$$ \eqalign{ (a+b)^4 + (a-b)^4 &= a^4 + 4a^3b + 6a^2b^2
                                    + 4ab^3 + b^4       \cr
                                &\qquad
                                 + a^4 - 4a^3b + 6a^2b^2
                                    - 4ab^3 + b^4       \cr
                                &= 2 ( a^4 + 6a^2b^2 +b^4) \cr}  $$
```

liefert

$$\begin{aligned}(a+b)^4 + (a-b)^4 &= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4 \\ &\qquad + a^4 - 4a^3b + 6a^2b^2 - 4ab^3 + b^4 \\ &= 2(a^4 + 6a^2b^2 + b^4)\end{aligned}$$

6 Tabellensatz

6.1 Tabulatoren

Die Erstellung schöner Tabellen gehört zu den interessantesten, aber auch schwierigsten Aufgaben im Satzbereich. Damit eine Tabelle 'schön' aussieht, ist der Abstand zwischen den Spaltenelementen passend zu wählen, sind bestimmte Teile durch andere Schriftarten hervorzuheben und die einzelnen Tabellenteile durch Linien zu gliedern. Beim Tabellensatz gibt es zwei verschiedene Aufgabenstellungen: auf der einen Seite liegen die Daten vor, und diese sollen als Tabelle gestaltet werden, auf der anderen liegt das Format als Vorgabe fest, etwa ein Fahrplan, und die Tabelle muß nur noch gefüllt werden.

Zum Satz von Tabellen gibt es in TEX zwei Methoden: Nutzung von 'Tabulatoren' und die automatische Ausrichtung der Tabelle mittels `\halign` — *horizontal alignment*. Beim Satz mittels Tabulatoren wird die Breite der Tabellenspalten *explizit* vorgegeben.

Die einfachste Form der Breitenangabe ist die Vorgabe der Spaltenzahl:
`\settabs 4 \columns` definiert Tabulatoren für vier Spalten. Die Aktivierung der Tabulatoren geschieht mittels `\+` ... `\cr`. Die Zeichen zwischen `\+` und `\cr` werden mittels Tabulatoren gesetzt. Das Tabulatorsymbol ist dabei '`&`'.

Die Eingabe

```
\settabs 4 \columns
\+erste Spalte &zweite Spalte &dritte Spalte  &vierte Spalte \cr
\+ eins & zwei & drei  & vier \cr
\+ one  & two  & three & four \cr
```

ergibt 4 Tabellenspalten.

Diese werden dann mit der Information gefüllt:

erste Spalte	zweite Spalte	dritte Spalte	vierte Spalte
eins	zwei	drei	vier
one	two	three	four

Aber auch:

```
\+erste Spalte &&&& zu weit! \cr
```

ergibt:

erste Spalte zu weit!

Im Gegensatz zum "TAB" einer normalen Schreibmaschine bewirken zum Beispiel zwei 'TABS' ('`&`') immer die Positionierung auf den Anfang der dritten Spalte — unabhängig davon, wie voll die ersten beiden Spalten schon sind. Eventuell kann es dadurch zum Überschreiben kommen, falls die Spalten eins und zwei bis in die dritte Spalte hinein gefüllt sind. Die Leerzeichen nach einem "`&`" werden übrigens überlesen.

Die zweite und meist praktischere Technik, die Spalten zu definieren, geschieht mittels Angabe einer *Musterzeile*. Nach `\settabs` folgt einfach eine normale Tabulierungszeile mit `\+` am Anfang, `&` für die Positionen und `\cr` am Ende. Die Information zwischen den einzelnen "`&`"s bestimmt dann die Breite der Spalten. Natürlich werden dann jeweils die breitesten Elemente mit zusätzlichem Leerplatz, etwa ein '`\quad`' oder '`\hskip 2cm`', angegeben.
Beispiel:

```
\settabs\+Sortierverfahren\quad  &      100 Elemente
             &      200 Elemente &      500 Elemente &\cr
\+Sortierverfahren & 100 Elemente & 200 Elemente
                           & 500 Elemente \cr
\+Austausch        & 250          & 1000   & 10000 \cr
\+Einf\"ugen       & 200          & 400    &  3000 \cr
\+Auswahl-Sort     & 110          & 260    &  2000 \cr
\+Shell-Sort       &  70          & 250    &   700 \cr
\+Heap-Sort        &  50          & 100    &   300 \cr
\+Quick-Sort       &  40          &  60    &   200 \cr
```

ergibt die Aufteilung

und die Ausgabe

Sortierverfahren	100 Elemente	200 Elemente	500 Elemente
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Diese Tabelle ist aber noch nicht besonders schön. Die Ausrichtung der Elemente in den Spalten sollte rechtsbündig sein. Durch Einfügung von `\hfill` *(dynamischer Leerplatz)*

erreicht man eine solche Ausrichtung.

```
\settabs\+Sortierverfahren\quad   & 100 Elemente
                   & 200 Elemente & 500 Elemente &\cr
\+Sortierverfahren & 100 Elemente
                   & 200 Elemente & 500 Elemente &\cr
\+Austausch        &\hfill 250  & \hfill 1000 & \hfill 10000 & \cr
\+Einf\"ugen       &\hfill 200  & \hfill 400  & \hfill  3000 & \cr
\+Auswahl-Sort     &\hfill 110  & \hfill 260  & \hfill  2000 & \cr
\+Shell-Sort       &\hfill 70   & \hfill 250  & \hfill   700 & \cr
\+Heap-Sort        &\hfill 50   & \hfill 100  & \hfill   300 & \cr
\+Quick-Sort       &\hfill 40   & \hfill 60   & \hfill   200 & \cr
```

ergibt

Sortierverfahren	100 Elemente	200 Elemente	500 Elemente
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Anmerkung: Damit das `\hfill` auch in der letzten Spalte wirkt, muß die Breite dieser Spalte bekannt sein. Daher wird noch ein zusätzliches '`&`' gesetzt.

Jedes Tabellenelement bildet für sich eine eigene Klammergruppe. Damit können beim Aufruf durchaus Anwahlen anderer Schriften verwendet werden, diese sind dann jeweils auf nur ein Element beschränkt. Die rechtsbündige Ausgabe wurde oben vorgestellt; zentriert wird, indem auf beiden Seiten eines Tabellenelements ein `\hfill` gesetzt wird.

Im folgenden Beispiel sollen die Überschriften und die Namen fett gedruckt und die Daten zentriert werden. Zwischen der Überschrift und dem folgenden Text wird mittels `\smallskip` noch ein wenig Leerraum gelassen.

```
\settabs\+\bf Sortierverfahren\quad   & \bf100 Elemente
                   & \bf 200 Elemente & \bf 500 Elemente &\cr
        \+\bf Sortierverfahren\quad   & \bf100 Elemente
                   & \bf 200 Elemente & \bf 500 Elemente &\cr
\smallskip
\+\bf Austausch     &\hfill 250   \hfill & \hfill 1000 \hfill &
                     \hfill 10000 \hfill & \cr
\+\bf Einf\"ugen    &\hfill 200   \hfill & \hfill 400  \hfill &
                     \hfill 3000  \hfill & \cr
...
```

Dies liefert:

Sortierverfahren	**100 Elemente**	**200 Elemente**	**500 Elemente**
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Der Schreibaufwand für eine solche Tabelle erscheint mit Recht als ziemlich umfangreich. Eine komfortable Technik, solche Tabellen auch automatisch zu setzen, wird im folgenden Abschnitt *Automatischer Tabellensatz* dargestellt.

Eine angenehme Methode, die Tabelle bei der Ausgabe zentriert und mit etwas Abstand nach oben und unten zu setzen, ist die Nutzung des Mathematiksatzes: Durch die Eingabe

```
$$\vbox{\settabs .... \cr}$$
```

wird zunächst in den mathematischen Formelsatz gewechselt, und zwar für zentriert hervorgehobene Formeln. Der Befehl `\vbox` verläßt den mathematischen Modus — später wird `\vbox` noch genauer erläutert — und der Rest wird normal gesetzt. Ein meist positiver Nebeneffekt dieser Angabe ist die Tatsache, daß die Tabelle über Seitengrenzen *nicht* umbrochen werden kann. Durch die `\vbox` kann sie nicht mehr geteilt werden. Die letzte Tabelle hat auf diese Weise die folgende Ausgabe:

Sortierverfahren	**100 Elemente**	**200 Elemente**	**500 Elemente**
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Noch einige Anmerkungen: `\settabs`, `\+` und `\cr` benutzen intern den im folgenden erläuterten `\halign`-Befehl. Jede Tabellenzeile ist im Prinzip eine Tabelle für sich ganz alleine. Damit wird die Tabelle, wenn sie nicht in einer `\vbox` verpackt ist, auch über Seitengrenzen umbrochen.

Auf zwei Unschönheiten sei auch noch hingewiesen:

— Aus einer Tabelle heraus sind keine Fußnoten möglich.

— Wird ein `\cr` vergessen, bewirkt dies eine ziemliche Verhedderung in der Verarbeitung der folgenden Eingabe. Das TEX-Programm meldet später zwar meist Fehler, nur die eigentliche Ursache ist nicht sofort erkenntlich.

6.2 Variable Tabulatoren

Gelegentlich werden Tabellen gesetzt, in denen sich die Tabulatorpositionen häufig ändern. Um etwa die folgende Trenntabelle zu setzen, wären eine Reihe von `\settabs`-Befehlen nötig, die die Eingabe äußerst unhandlich gestalten würden.

ta-bel-la-risch
-le
-len-art
-kopf
-satz
-bu
-la-tor
-cho
-me-ter

Es gibt nun einen Löschbefehl für Tabulatoren — ähnlich wie auf einer Schreibmaschine — zur Entfernung aller rechts stehenden Tabulatoren. Auf der anderen Seite kann in einer Tabulierungseingabe (`\+...\cr`) dynamisch ein neuer Tabulator gesetzt werden. Wird ein "`&`" eingegeben, so wird damit ein neuer Tabulator gesetzt. Dies funktioniert aber nur, wenn rechts von der aktuellen Position keine Tabulatoren gesetzt sind.
Die Eingabe für die obige Tabelle hat die folgende Gestalt:

```
\+\cleartabs \bf ta&-bel&-la&-risch&\cr
\+                  &    &\cleartabs-le&\cr
\+                  &    &\cleartabs-len&-art&\cr
\+                  &    &              &\cleartabs-kopf&\cr
\+                  &    &              &\cleartabs-satz&\cr
\+                  &\cleartabs-bu&\cr
\+                  &             &\cleartabs-la&-tor&\cr
\+                  &\cleartabs-cho&\cr
\+                  &              &\cleartabs-me&-ter\cr
```

Zu Beginn eines neues Eintrags, dort wo sich bei den Tabulatoren etwas verändert, steht jeweils ein `\cleartabs`-Befehl. Dieser löscht dann alle folgenden Tabulatoren. Anschließend werden jeweils durch "`&`" neue Tabulatoren gesetzt.

Durch eine einfache Definition `\def\ct{\cleartabs}` kann anstelle von `\cleartabs` ein "`\ct`" geschrieben werden, welches den Schreibaufwand natürlich reduziert.

6.3 Automatischer Tabellensatz: die Musterzeile

Wirklichen Komfort beim automatischen Tabellensatz erreicht man erst durch den Befehl `\halign`. Dabei wird die gesamte Information, die zu einer Tabelle gehört, eingelesen und aus ihr die nötige Spaltenbreite bestimmt. Das größte Element in jeder Spalte legt dann jeweils die Spaltenbreite fest.
`\halign` arbeitet nach folgender Eingabesyntax:

```
\halign{  Musterzeile \cr Zeile_1 \cr
                          Zeile_2 \cr
                             ...
                          Zeile_n  \cr }
```

Die wichtigsten Elemente der Musterzeile sind "`#`" und "`&`". Dabei trennt `&` — wie bei `\settabs` — die einzelnen Spalten. '`#`' ist der Platzhalter für die Stelle, an die das Tabellenelement eingesetzt werden soll. Der Text links von "`#`" wird jeweils vor das Element, der Text rechts von "`#`" bis zum "`&`" hinter das Element eingefügt. Dies soll zunächst an unserem bekannten Beispiel demonstriert werden:

```
\halign { \bf#  \quad & \hfill # \quad &
                        \hfill # \quad & \hfill #     \cr
 %%%%%%%%
 %%%%%%%% <-Spalte 1-> <--Spalte 2---->
 %%%%%%%%              <--- Spalte 3--> <--Spalte 4-->
 %%%%%%%%
   Sortierverfahren & 100 Elemente & 200 Elemente & 500 Elemente \cr
\noalign{\smallskip}
   Austausch        & 250          & 1000   & 10000 \cr
   Einf\"ugen       & 200          & 400    &  3000 \cr
   Auswahl-Sort     & 110          & 260    &  2000 \cr
   Shell-Sort       &  70          & 250    &   700 \cr
   Heap-Sort        &  50          & 100    &   300 \cr
   Quick-Sort       &  40          &  60    &   200 \cr   } % Ende
```

Dies ergibt:

Sortierverfahren	100 Elemente	200 Elemente	500 Elemente
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Dazu einige Anmerkungen: In der ersten Spalte wird automatisch in die Schrift **boldface** umgeschaltet. Die hinteren Spalten werden automatisch rechtsbündig gesetzt, indem vor `#` ein `\hfill` steht. Durch `\noalign{ ... }` wird Information eingefügt, die nicht in den Satz eines Tabellen*elementes* berücksichtigt wird. Typisch ist hierbei vertikaler Skip für zusätzlichen Abstand. In dem oberen Beispiel wird einmal der Befehl `\noalign{\smallskip}` nach der Überschrift verwendet.

Das zunächst Abschreckende an der Anwendung von `\halign` ist die merkwürdig erscheinende Musterzeile. An dieses Konzept hat man sich erst einmal zu gewöhnen. Die beiden wichtigsten Elemente der Musterzeile sind "`#`" für den Platzhalter und "`&`" als Spaltentrenner. Vor dem die Musterzeile beendenden "`\cr`" darf als letztes Element nur ein "`#`" auftreten, kein "`&`". Es darf also keine unvollendete Spalte begonnen werden, sonst wird der Fehler "`missing #`" gemeldet.
Die einfachste Form der Musterzeile ist in der folgenden Anweisung verwendet:

```
\halign{#&#&#&#\cr ...}
```

Dies erzeugt eine Tabelle mit vier Spalten, bei denen jede so breit ist wie ihr entsprechend größtes Element. Die Elemente werden, jedes in seiner Spalte, linksbündig

nebeneinander gesetzt. Die Ausgabe erfolgt dabei ohne weiteren Leerraum, da weder in der Musterzeile noch in den Eingabezeilen Leerraum vorgegeben wird.

```
\halign{#&#&#&#\cr
eins&zwei&drei&vier\cr
f\"unf&sechs&sieben&acht\cr
neun&zehn&elf&zw\"olf\cr}
```

erzeugt die Tabelle:

eins	zwei	drei	vier
fünf	sechssiebenacht		
neunzehn elf			zwölf

Durch Leerzeichen kann die Eingabe besser gegliedert werden. Insbesondere wird dann auch zwischen den einzelnen Spalten mindestens der Leerraum eines Leerzeichens gelassen. Mehrere Leerzeichen hintereinander wirken dabei wie ein Leerzeichen. Die Eingabe

```
\halign{#&#&#&#\cr
eins   & zwei  & drei   & vier    \cr
f\"unf & sechs & sieben & acht    \cr
neun   & zehn  & elf    & zw\"olf \cr}
```

liefert das folgende Ergebnis:

eins	zwei	drei	vier
fünf	sechs	sieben	acht
neun	zehn	elf	zwölf

Es können nun spaltenspezifische Einstellungen in der Musterzeile gesetzt werden. Alle Eingaben in einem Musterelement sind beschränkt auf dieses Element. (Die einzige Ausnahme ist `\tabskip`!) Im folgenden Beispiel wird die erste Spalte in **boldface** und die dritte in *italic* und die letzte in `typewriter` gesetzt.

```
\halign{\bf #&#&\it #&\tt #\cr
%       ^^^     ^^^^  ^^^^
%       (1)     (3)   (4)
%
     eins   & zwei  & drei   & vier    \cr
     f\"unf & sechs & sieben & acht    \cr
     neun   & zehn  & elf    & zw\"olf \cr}
```

liefert

eins	zwei	*drei*	`vier`
fünf	sechs	*sieben*	`acht`
neun	zehn	*elf*	`zwölf`

Selbst kompliziertere Konstruktionen mit einer automatischen Anwahl des Mathematikmodus sind möglich:

```
\halign{\it #\quad & $#$ \cr
   Minimum bei & \sqrt{\pi}   \cr
   Maximum bei & \sqrt{\pi+1} \cr}
```

liefert

Minimum bei	$\sqrt{\pi}$
Maximum bei	$\sqrt{\pi+1}$

Allerdings kann es vorkommen, daß die Musterzeile mit ihren konstanten Einstellungen fast immer paßt, aber eben nur fast immer. Bei nur wenigen Elementen möchte man auf die Vorgabe des Musterelementes verzichten. Hier hilft der Befehl `\omit`, der — am Anfang eines Tabellenelementes gesetzt — die Vorgabe durch das Musterelement unterdrückt. Im letzten Beispiel wird in der zweiten Spalte automatisch in den Mathematiksatz gewechselt. In der folgenden Eingabe soll dies bei dem letzten Eintrag nicht geschehen:

```
\halign{\it #\quad & $#$ \cr
   Minimum bei  & \sqrt{\pi}   \cr
   Maximum bei  & \sqrt{\pi+1} \cr
   Unstetigkeit &\omit \it nicht gefunden \cr}
```

Minimum bei	$\sqrt{\pi}$
Maximum bei	$\sqrt{\pi+1}$
Unstetigkeit	*nicht gefunden*

Hier wird im letzten Element ein "`\omit`" gesetzt, dies reduziert den Mustereintrag auf ein "`#`".

6.4 Automatischer Tabellensatz: Spaltenausrichtung

Die typischen Einstellungen, die für den Satz einer Tabelle vollzogen werden, sind:

Ausrichtung:	links, rechts, zentriert
Schriftanwahl:	`\bf, \it, \sl ...`
Spaltenabstand:	links, rechts
Einfügung konstanter Textteile	

Dabei sind die Ausrichtung und die Bestimmung des Spaltenabstands der schwierigere Teil:

Ein Standardfall beim Tabellensatz ist die Ausgabe von Zahlenkolonnen, die nach dem Dezimalpunkt oder Dezimalkomma ausgerichtet sind. Dies wird erst dann problematisch, wenn nicht alle Zahlen die gleiche Anzahl Vorkommastellen beziehungsweise Nachkommastellen besitzen. Hier bietet die Verwendung von zwei Tabellenspalten, von denen die erste rechtsbündig und die zweite linksbündig gesetzt wird, einen Ausweg. An Stelle des Kommas ist dann ein "`&`" einzugeben. Der Mathematikmodus wird verwendet, um ein mathematisches Minus "−" und keinen Trennstrich "-" zu erhalten.

Beispiel:

```
$$\vbox{\halign{ #\quad &\hfill#,&#\hfill\cr
        $ \sin    15^\circ   $   &  $0$&259 \cr
        $ \sin    30^\circ   $   &  $0$&5   \cr
        $ \sin    45^\circ   $   &  $0$&707 \cr
        $ \sin {-15}^\circ   $   & $-0$&259 \cr
        $ \sin {-30}^\circ   $   & $-0$&5   \cr}}$$
```

liefert

$$\begin{array}{ll} \sin 15^\circ & 0{,}259 \\ \sin 30^\circ & 0{,}5 \\ \sin 45^\circ & 0{,}707 \\ \sin -15^\circ & -0{,}259 \\ \sin -30^\circ & -0{,}5 \end{array}$$

Übrigens ist die Eingabe ökonomischer, wenn die konstanten Teile alle in die Musterzeile gezogen werden:

```
$$\vbox{\halign{ $\sin{#}^\circ$\quad &\hfill$#,$&$#$\hfill\cr
              15       &  0&259 \cr
              30       &  0&5   \cr
              45       &  0&707 \cr
             -15       & -0&259 \cr
             -30       & -0&5   \cr}}$$
```

6.5 Automatischer Tabellensatz: Spaltenabstand

Häufig möchte man Tabellen auf eine bestimmte Breite bringen, indem die Spalten mit Leerraum gleichmäßig aufgefüllt werden. Die Steuergröße dafür ist die TEX-Variable `\tabskip`. Durch `\tabskip` bestimmter Leerraum wird bei einer Tabelle aus vier Spalten an folgenden Positionen gesetzt:

(A) ␣ (1) ␣ (2) ␣ (3) ␣ (E)

Dabei sind die Position "(A)" und "(E)" der Leerraum vor der ersten Spalte und nach der letzten Spalte. In der folgenden `\halign`-Anweisung sind die Positionen markiert, die anzeigen, welcher `\tabskip`-Wert zu welchem Zeitpunkt benutzt wird:

(A) `\halign{..#..` (1) `&..#..` (2) `&..#..` (3) `&..#..` (E) `\cr`

Das heißt, für den Leerraum *vor* der Tabelle wird der Wert verwendet, der zum Zeitpunkt von `\halign` gültig ist, für den Leerraum *nach* der letzten Spalte der Wert, der zum Zeitpunkt des die Musterzeile beendenden `\cr` gültig ist. Der Leerraum zwischen den Spalten regelt sich jeweils zum Zeitpunkt des "`&`".
An der folgenden Beispieleingabe soll die Verwendung von `\tabskip` demonstriert wer-

den:

```
$$\vbox{\halign{#&#&#\cr % 3 Spalten
    \it PLZ  & \it Ort      & \it Vorwahl \cr
        1000 & Berlin       & (030)  \cr
        2000 & Hamburg      & (040)  \cr
        3000 & Hannover     & (0511) \cr
        4000 & D\"usseldorf & (0211) \cr
        5000 & K\"oln       & (0221) \cr
        6000 & Frankfurt    & (069)  \cr
        7000 & Stuttgart    & (0711) \cr
        8000 & M\"unchen    & (089)  \cr}}$$
```

Diese Eingabe erzeugt die folgende Ausgabe, wobei der Abstand zwischen den Spalten aus den Leerzeichen bei der Eingabe herrührt:

PLZ	*Ort*	*Vorwahl*
1000	Berlin	(030)
2000	Hamburg	(040)
3000	Hannover	(0511)
4000	Düsseldorf	(0211)
5000	Köln	(0221)
6000	Frankfurt	(069)
7000	Stuttgart	(0711)
8000	München	(089)

Diese Leerzeichen werden bei der Berechnung der Spaltenbreiten berücksichtigt. Leerzeichen in der Musterzeile hätten übrigens die gleiche Wirkung.

In der folgenden Variante wird ein `\tabskip=15pt` ($\approx$ 5 mm) gesetzt. Die Eingabe wird in der ersten Zeile verändert:

```
$$\vbox{\tabskip=15pt \halign{#&#&#\cr % 3 Spalten
```

Dadurch erhält man folgendes Aussehen:

PLZ	*Ort*	*Vorwahl*
1000	Berlin	(030)
2000	Hamburg	(040)
3000	Hannover	(0511)
4000	Düsseldorf	(0211)
5000	Köln	(0221)
6000	Frankfurt	(069)
7000	Stuttgart	(0711)
8000	München	(089)

Der `\tabskip`-Wert braucht auch nicht in allen Elementen gleich groß zu sein. Er kann in einem Musterelement verändert werden und gilt dann für die folgenden weiter.

Elegant wird die Verwendung von `\tabskip` und `\halign`, wenn beim `\halign`-Befehl die gewünschte Breite der zu setzenden Tabelle angegeben wird. Die Breitenangabe geschieht durch `\halign to <`*dimension*`>`. Beispiele:

```
\halign to 200pt{...}
\halign to 15cm{...}
\halign to 0.8\hsize{...}
```

Es ist auch eine Breitenangabe möglich, die die zusätzliche Ausdehnung der Tabelle festlegt. Damit wird also der Leerraum definiert, der bei geeignetem `\tabskip` gleichmäßig verteilt wird. Beispiel:

```
\halign spread 4cm{..}
```

Um die Tabelle dann überhaupt auffüllen zu können, ist der `\tabskip` mit einem "dynamischen" Anteil zu definieren, das heißt mit einer Längenangabe, um die das einzelne Spaltenelement vergrößert werden darf. Der so angegebene "`plus`"-Anteil ist die 'Füllmasse' für die Tabelle.

```
\tabskip=15pt plus 200pt
```

erlaubt die Auffüllung in jedem Spaltenelement um `200 pt`. Die letzte Tabelle kann so über die ganze Zeile gedehnt werden.

```
$$\vbox{\tabskip=15pt plus 200pt\halign to \hsize{...
%                     ^^^^^^^^^^        ^^^^^^^^^
```

liefert die folgende etwas gekürzte Tabelle

PLZ	*Ort*	*Vorwahl*
1000	Berlin	(030)
2000	Hamburg	(040)
3000	Hannover	(0511)
4000	Düsseldorf	(0211)
5000	Köln	(0221)

Durch

```
\tabskip=0pt plus 20cm
\halign to \hsize{\tabskip=15pt#&#&#
                  \tabskip=0pt plus 20cm \cr
```

erhält man eine Zentrierung mittels `\tabskip`. Zwischen die Spalten wird jeweils 15 pt Leerraum zusätzlich gesetzt. Da die Tabelle schon die Breite von `\hsize` besitzt, ist die Zentrierung mittels `$$\vbox...` entfernt worden. Das Ergebnis zeigt dann keinen zusätzlichen vertikalen Leerraum vor und nach der Tabelle:

PLZ	*Ort*	*Vorwahl*
1000	Berlin	(030)
2000	Hamburg	(040)
3000	Hannover	(0511)
4000	Düsseldorf	(0211)
5000	Köln	(0221)

6.6 Automatischer Tabellensatz: gerahmte Tabellen

Sehr beliebt sind Tabellen in Kästchen: Die waagrechten Linien sind dabei der einfache Teil: `\noalign{\hrule}` zieht eine passend lange waagrechte Linie. Die senkrechten Linien werden Stück für Stück aus Einzelelementen jeder Zeile zusammengesetzt. Ein senkrechtes Linienelement heißt `\vrule`. Damit diese Elemente auch untereinander stehen und den Eindruck einer geschlossenen Linie bilden, sollte man extra Tabellenspalten für die senkrechten Striche einführen.

Beispiel:

```
{\offinterlineskip \tabskip=0pt % wegen senkrechter Linien
\halign{ \strut                 % wegen Unterlängen
         \vrule#&               % Spalte 1  -- senkrechte Linie
 \quad   \bf#  \quad &          % Spalte 2  -- Text in boldface
         \vrule#&               % Spalte 3  -- senkrechte Linie
         \quad                  % Spalte 4  -- Text rechtsbündig
         \hfil # \quad &        %
         \vrule#&               % Spalte 5  -- senkrechte Linie
         \quad                  % Spalte 6  -- Text rechtsbündig
         \hfil # \quad &        %
         \vrule#&               % Spalte 7  -- senkrechte Linie
         \quad                  % Spalte 8  -- Text rechtsbündig
         \hfil # \quad &        %
         \vrule#                % Spalte 9  -- senkrechte Linie
         \cr                    % ENDE der Musterzeile
   \noalign{\hrule}
 & Sortierverfahren && 100 Elemente && 200 Elemente
                   && 500 Elemente &\cr
   \noalign{\hrule}
 & Austausch        && 250          && 1000  && 10000 & \cr
 & Einf\"ugen       && 200          && 400   &&  3000 & \cr
 & Auswahl-Sort     && 110          && 260   &&  2000 & \cr
 & Shell-Sort       && 70           && 250   &&   700 & \cr
 & Heap-Sort        && 50           && 100   &&   300 & \cr
 & Quick-Sort       && 40           && 60    &&  200  & \cr
   \noalign{\hrule} }                              } % ENDE
```

ergibt:

Sortierverfahren	100 Elemente	200 Elemente	500 Elemente
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Dabei sind noch einige Befehle zu erklären: `\offinterlineskip` setzt den zusätzlichen Leerplatz, den TEX zwischen einzelnen Zeilen ausgibt, auf Null, da sonst die senkrechten Linien nicht aneinanderstoßen.

Es gibt noch einige Steuerungsmöglichkeiten, die bisher nicht benutzt wurden, aber in einigen Fällen interessant sind. Hier ist eine Zusammenfassung:

`\offinterlineskip` schaltet den zusätzlichen Platz zwischen Zeilen ab — wichtig bei Tabellen mit 'Rähmchen'.

`\hidewidth` Wird dieser Befehl an den Anfang eines Tabellenelementes — hinter `&` — gesetzt, so wird die Größe dieses Elementes *nicht* bei der Kalkulation der nötigen Spaltenbreite berücksichtigt. Das Element kann also durchaus in die Nachbarspalten hineinragen. Diese sollten dann möglichst leer sein, sonst kommt es zum Übereinanderdrucken. Im Gegensatz zu `\multispan` werden die Definitionen aus der Musterzeile berücksichtigt.

`\strut` Sorgt für virtuell gleich hohe Zeilen. Praktisch wird ein unsichtbarer senkrechter Strich gesetzt, der die Maximalhöhe und Maximaltiefe eines Zeichens mit Unterlängen hat. Dies soll näher erläutert werden: An sich ist eine Zeile nur so hoch wie ihr größtes Element. Eine Zeile enthalte nun nur Zeichen wie 'x', 'u' oder 'n'. Ohne Ober- und Unterlängen ist diese Zeile nur sehr dünn. Der Befehl `\vrule` erzeugt nun einen senkrechten Strich, der so hoch wie die interne Höhe der Zeile ist. Die Anweisung `\offinterlineskip` wiederum hat den Standardzeilenabstand mittels `\baselineskip` abgeschaltet, so daß diese Zeile nun auch dicht an die vorangehende gesetzt wird. Das gleichmäßige Aussehen und der gleichmäßige Zeilenabstand wird dann wieder durch `\strut` erzwungen.

`\omit` Dies bewirkt, an den Anfang eines Tabellenelementes gesetzt, daß das zugehörige Musterelement auf '`#`' reduziert wird. Damit kann man z.B. ein `\vrule`, das in der Musterzeile steht, in einzelnen Positionen entfallen lassen.

`\multispan n` An den Anfang eines Elementes gesetzt, teilt TEX mit, daß die nachfolgende Information über `n` Tabellenspalten reichen soll — für Überschriften! Für `n` ist die Anzahl der Spalten anzugeben. Die Musterelemente werden wie bei `\omit` *nicht* berücksichtigt.

`&&` Wird in der Musterzeile eine Spalte nicht mit einem '`&`', sondern mit '`&&`' eingeleitet, so bedeutet dies, daß die folgenden Spalten beliebig oft wiederholt werden — je nach Bedarf.

Typische Fehlersituationen

Hier seien noch zwei typische Fehlersitutationen für gerahmte Tabellen dargestellt:

	eins	zwei	drei	vier
	fünf	sechs	sieben	acht
	neun	zehn	elf	zwölf

eins	zwei	drei	vier
fünf	sechs	sieben	acht
neun	zehn	elf	zwölf

In der linken Tabelle ist der `\tabskip` zu Beginn und zum Ende der Musterzeile nicht Null, dadurch wird die Tabelle breiter und mit Leerraum zum Anfang und Ende gefüllt. Dieser Leerraum wird auch durch das `\noalign{\hrule}` mitberücksichtigt.

In der rechten Tabelle fehlen das `\offinterlineskip` und die für Unter- und Oberlängen nötigen `\strut`-Befehle.

Auf ein technisches Problem sei bei gerahmten Tabellen im Zusammenhang mit `\noalign` hingewiesen: Bei Tabellen ohne senkrechte Striche kann man zusätzlichen vertikalen Leerraum mittels `\noalign{\smallskip}` oder ähnlichen Befehlen zwischen den einzelnen Zeilen lassen. Dies ist hier allerdings nicht möglich, da einfach Löcher in den senkrechten Linien entstehen. Es gibt zwei Wege, dieses Problem zu umgehen:

Im ersten Fall fügt man leere Tabellenzeilen von der Struktur “ `& & & \cr` ” ein. Damit erhält man leere Tabellenzeilen. Dies setzt allerdings voraus, daß neben den senkrechten Linien keine konstanten Texte ausgegeben werden. Wird der Abstand zu groß, kann zum Beispiel mittels `\noalign{\vskip-0.5\baselineskip}` um eine halbe Zeilenhöhe zurückgegangen werden. Die senkrechten Linien überlappen sich dann, dies ist aber unschädlich.

Im zweiten Fall wird eine Tabellenzeile mit einem etwas überhöhten Element gesetzt. Dazu muß man wissen, daß die normale Zeile eine Höhe von 8.5 pt und eine Tiefe von 3.5 pt besitzt. Soll ein Abstand entsprechend “`\smallskip`” vor eine aktuelle Tabellenzeile gesetzt werden, kann man also durch den Befehl

```
\vrule height 11.5pt width 0pt
```

die Zeilenhöhe vergrößern. Soll das “`\smallskip`” nach der aktuellen Zeile erzeugt werden, hilft

```
\vrule depth 6.5pt width 0pt
```

Hier sei noch eine ‘Luxusausgabe’ der Beispieltabelle dargestellt:

Sortierverfahren	Anzahl sortierter Elemente		
	100	200	500
Austausch	250	1000	10000
Einfügen	200	400	3000
Auswahl-Sort	110	260	2000
Shell-Sort	70	250	700
Heap-Sort	50	100	300
Quick-Sort	40	60	200

Bei dieser Tabelle sind folgende Besonderheiten im Satz verwendet worden: Die umrahmenden Linien besitzen eine größere Strichstärke als die Linien im Innern. Ein Titelelement wird über mehrere Spalten gleichzeitig gesetzt. In der ersten Spalte erfolgt die automatische Schriftanwahl “`\it`”, die Zahlenspalten werden rechtsbündig gesetzt. Die externe Tabellenbreite von `0.8\hsize` wird durch einen dynamischen `\tabskip` in der ersten Tabellenspalte erreicht. Das Wort “Sortierverfahren” wird durch die Kombination der Befehle `\smash` und `\lower` ein wenig tiefer gesetzt, ohne daß die Höhe und die Tiefe der ersten Titelzeile beeinflußt werden. Der Befehl `\lower` versetzt die nachfolgende Box nach unten um den angegebenen Betrag. Der Befehl `\unskip` in der Musterzeile entfernt jeweils den Leerraum, der durch die nachlaufenden Leerzeichen in den Tabelleneinträgen erzeugt wird.

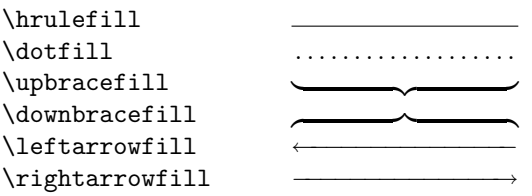
Die Eingabedaten für diese Tabelle lauten:

```
$$                                          % zum Zentrieren
 \offinterlineskip                          % kein automatischer Leerraum
                                            %     zwischen den Zeilen
 \tabskip=0pt                               % äußerer Tabskip=0
 \vbox{                                     % Math. Mode verlassen
 \halign to 0.8\hsize                       % Tabellenbreite 80 %
                                            %     einer Zeile
   {\strut                                  % Höhenausgleich
    \vrule width0.8pt\quad#                 % Sp. 1: dicke Linie
    \tabskip=0pt plus1000pt                 %        \tabskip für
                                            %        diese Spalte
  & \it#  \quad                             % Sp. 2: italic
  & \vrule#                                 % Sp. 3: Linie
    \tabskip=0pt                            % für den Rest der Tabelle
  & \quad \hfil #\unskip \quad               % Sp. 4: rechtsbündig
  & \vrule#                                 % Sp. 5: Linie
  & \quad \hfil #\unskip \quad               % Sp. 6: rechtsbündig
  & \vrule#                                 % Sp. 7: Linie
  & \quad \hfil #\unskip \quad               % Sp. 8: rechtsbündig
  & \vrule width0.8pt#                      % Sp. 9: dicke Linie
    \cr                                     % MUSTER-ENDE  %%%%%%%%
                                            %
  \noalign{\hrule}\noalign{\hrule}          % waagrechte dicke Linie
                                            %
                                            % 1.Titelzeile
  & \smash{\lower6pt\hbox                   %  a)  "Sortierverfahren"
           {\bf Sortierverfahren}}          %      tiefergestellt
      &&\multispan 5 \hfill                  %  b)  \multispan 5
     Anzahl sortierter Elemente             %      => 5 Spalten zusammen-
                           \hfill & \cr      %         gefaßt (zentriert)
    &&&      100 &\omit& 200                % 2.Titelzeile
                 &\omit&500 & \cr           %   \omit => ohne Striche
      \noalign{\hrule}                      % Zwei waagrechte Striche
      \noalign{\hrule}                      %   ergeben einen dicken Strich.
                                            %
  & Austausch    && 250 && 1000 && 10000 & \cr
  & Einf\"ugen   && 200 && 400  && 3000  & \cr
  & Auswahl-Sort && 110 && 260  && 2000  & \cr
  & Shell-Sort   && 70  && 250  && 700   & \cr
  & Heap-Sort    && 50  && 100  && 300   & \cr
  & Quick-Sort   && 40  && 60   && 200   & \cr
                                            %
   \noalign{\hrule\hrule}                   % dicker waag. Strich
}}                                          % Ende \halign\vbox
$$
```

6.7 Hilfsmittel beim Tabellensatz

Spaltenlinien

Einige Befehle sind im Zusammenhang mit dem Satz von Tabellen sehr interessant. Sie können sehr schön zur optischen Aufbereitung komplexer Tabellen verwendet werden. Zunächst einmal gibt es die Gruppe von Befehlen, die ein Spaltenelement mit einem Muster bis zu seiner gewünschten Breite auffüllen:



Mit Hilfe dieser Befehle wird jeweils ein Element bis zu seiner geforderten Breite aufgefüllt. Dies sei am Beispiel von `\dotfill` dargestellt.

```
$$\vbox{\tabskip=40pt
        \halign{#\dotfill\quad&(#\unskip) \cr
         Berlin               &  030   \cr
         Hamburg              &  040   \cr
         D\"usseldorf-Zentrum &  0211  \cr}}$$
```

liefert die folgende Ausgabe. `\unskip` entfernt dabei den Leerraum, der durch die Leerzeichen zwischen den Zahlen und `\cr` erzeugt würde.

Berlin	(030)
Hamburg	(040)
Düsseldorf-Zentrum	(0211)

Angewendet in einem — etwas überfrachteten — Beispiel:

```
$$\vbox{\offinterlineskip
        \tabskip=0pt
        \halign{\vrule\vrule\strut\quad $#^\circ$ \quad &
                \vrule#&
                \quad\hfill$#,$&
                #\hfill\quad\vrule\vrule\cr
        \noalign{\hrule\hrule}
        \sin{-15} && -0&259 \cr
        \omit\vrule\vrule&&\multispan2\dotfill\vrule\vrule \cr
        \sin{0}   &&  0&0   \cr
        \omit\vrule&&\multispan2\hrulefill \cr
        \cos{0}   &&  1&0   \cr
        \omit\vrule\vrule&&\multispan2\dotfill\vrule\vrule \cr
        \cos{15}  &&  0&966 \cr
        \noalign{\hrule\hrule}}}$$
```

liefert

$\sin -15^\circ$	$-0{,}259$
$\sin 0^\circ$	$0{,}0$
$\cos 0^\circ$	$1{,}0$
$\cos 15^\circ$	$0{,}966$

Hierbei werden durch doppelte `\hrule` und `\vrule`-Befehle dicke Linien erzeugt. Eine äquivalente Eingabe dazu ist

```
\vrule width 0.8pt
\hrule height 0.8pt
```

an den jeweils nötigen Stellen. Dabei wird die Tatsache verwendet, daß die Standardstrichdicke gerade 0.4 pt beträgt.

Dynamische endlose Musterzeilen

Wird in einer Musterzeile statt eines "`&`" gleich "`&&`" eingegeben, so bedeutet dies, daß die nachfolgenden Einträge bei Bedarf immer weiter wiederholt werden. Bei

```
\halign{#&&#\cr
%         ^^
```

wird das Musterelement "`&#`" beliebig oft implizit wiederholt. Bei

```
\halign{#&&\quad\hfill#&#\quad\cr
```

besteht das zu wiederholende Doppelelement aus "`&\quad\hfill#&#\quad`". Beginnt eine Musterzeile `\halign{&...}` gar mit einem "`&`", so wird der gesamte Eintrag wiederholt.
Im folgenden Beispiel sei die Anwendung veranschaulicht.

```
$$\vbox{\halign{&\hfill#\hfill\quad\cr
     $ x = $ &  1 &  2  &  3  & 4  & 5   & 6   & \dots \cr
     $ x!= $ &  1 &  2  &  6  & 24 & 120 & 720 & \dots \cr}}$$
```

liefert

$$\begin{array}{rccccccc} x = & 1 & 2 & 3 & 4 & 5 & 6 & \dots \\ x! = & 1 & 2 & 6 & 24 & 120 & 720 & \dots \end{array}$$

Erweiterung der Gesamttabellenbreite

Durch `\halign spread` <*dimension*> `{...}` wird die Tabelle in ihrer natürlichen Breite *plus* einem Zuwachs gesetzt, um den die gesamte Tabelle auseinandergezogen wird. Der Zuwachsanteil wird dann gemäß `\tabskip` verteilt.

Zusammengefaßte Spalten

Wird an Stelle von "`&`" ein "`\span`" in der Tabellenzeile gesetzt, so werden das linke und rechte Tabellenelement zusammengezogen und inklusive des dazwischenliegenden "`\tabskip`" gemeinsam gesetzt. Dabei werden die Musterelemente aber im Gegensatz

zu `\multispan` ausgewertet. Dieses etwas spröde klingende Verhalten sei an dem folgenden Beispiel veranschaulicht.

```
$$\hbox{\vrule\vbox
   {\tabskip=0pt
    \halign{\hfill#&#&\hfill#\cr    % 1. Spalte rechtsbündig
                                     % 2. Spalte unverändert
                                     %        (linksbündig \hfil)
                                     % 3. Spalte rechtsbündig
%     =================   =================   ==================
      \quad Links \quad & \quad Mitte \quad & \quad Rechts \quad \cr
         1000  \quad    &       2000        &         3000 \quad \cr
          100  \quad  \span      200        &          300 \quad \cr
           10  \quad    &         20      \span         30 \quad \cr
            1  \quad  \span        2      \span          3 \quad \cr
%     =================   =================   ==================
      }}\vrule}$$
```

liefert

Links	Mitte	Rechts
1000	2000	3000
	100 200	300
10	20	30
	1 2	3

Dabei werden in den einzelnen Zeilen — wie nachfolgendes Schema zeigt — Elemente zusammengezogen:

Links	Mitte	Rechts
.....		
	.....	
.....		

```
\hfill 100 \quad  200                    % Zeile 3  Element: 1 + 2
        20 \hfill  30  \quad             % Zeile 4  Element:     2 + 3
\hfill   1 \quad    2  \hfill 3 \quad    % Zeile 5  Element: 1 + 2 + 3
```

Also wird in Zeile 3 die Eingabe der Spalten 1 und 2 durch ein Leerzeichen getrennt rechtsbündig in der Doppelspalte 1/2 gesetzt.

In der Zeile 4 wird das Doppelelement aus den Spalten 2 und 3 gebildet. Dort wird durch ein zwischen die beiden Eingabeelemente gesetztes `\hfill` eine linksbündige Ausgabe für `20` und eine rechtsbündige für `30` erzeugt.

Zum Abschluß wird in der letzten Zeile eine zentrierte Ausgabe der beiden ersten Eingabeelemente und eine rechtsbündige Darstellung für das letzte Element erzeugt, wobei das Ausgabeelement aus allen drei Spalten besteht.

Ein `\span`-Befehl in der Musterzeile bewirkt übrigens eine Expandierung der nachfolgenden Anweisung. Damit wird ein folgender Makroaufruf sofort ausgewertet und nicht erst beim Einsetzen der Tabellenelemente.

7 Eigene Definitionen und Befehle — Makros

7.1 Einfache Makros

Eine wichtige Arbeitserleichterung ist die Abkürzung von häufig eingegebenen Textstücken durch *Makros*. Wenn in einem Text zum Beispiel häufig der Vektor $(x_1, \ldots, x_n)$ gesetzt werden muß, ist es schon sehr lästig, stets `$(x_1,\ldots,x_n)$` einzugeben. Der Makrobefehl `\def` hilft dabei:

```
\def\XV{(x_1,\ldots,x_n)}
```

bewirkt, daß der Aufruf `\XV` die *Abkürzung* dieses längeren Ausdrucks ist. Die Ersparnis verdeutlicht folgendes Beispiel Um die Formel

$$\sum_{(x_1,\ldots,x_n)\ne0}\bigl(f(x_1,\ldots,x_n)+g(x_1,\ldots,x_n)\bigr)$$

zu setzen, lautet die abgekürzte Schreibweise:

```
\def\XV{(x_1,\ldots,x_n)}
$$\sum_{\XV\ne0}\bigl(f\XV+g\XV\bigr)$$
```

Dagegen lautet die Eingabe ohne die Verwendung eines Abkürzungsmakros:

```
$$\sum_{(x_1,\ldots,x_n)\ne0}
    \bigl(f(x_1,\ldots,x_n)+g(x_1,\ldots,x_n)\bigr)$$
```

Wenn der Ausdruck `\XV` von TEX in der Eingabe gefunden wird, expandiert TEX dies als die dazugehörige Zeichenfolge `(x_1,\ldots,x_n)`. Die Wirkung ist genauso, als wenn diese Zeichen in der Eingabe stehen würden.

> *Die geschweiften Klammern, die um den Definitionstext herumstehen, werden nicht eingesetzt. Die nur zur Definition nötigen äußeren Klammern werden entfernt.*

Die beiden Definitionen

```
\def\zb{\bf zum Beispiel}
\def\ZB{{\bf zum Beispiel}}
```

unterscheiden sich in dem wichtigen Punkt, daß beim Aufruf des ersten Makros `\zb` auch noch nach dem Text '`zum Beispiel`' weiter in der Schrift `\bf` (boldface) gesetzt wird. Der Aufruf `\zb` wird als "`\bf zum Beispiel`" ausgewertet, dagegen `\ZB` als "`{\bf zum Beispiel}`".

> *Wenn notwendig, immer noch zusätzliche Blockklammern setzen. Im Normalfall sind diese unschädlich!*

7.2 Makros mit Parametern

Am häufigsten werden Makros angewendet, die Parameter besitzen. Will man ein allgemeines Makro für *x,y,z*-Vektoren aus dem obigen Beispiel, so empfiehlt sich:

```
\def\vektor#1{(#1_1,\ldots,#1_n)}/
```

Dann liefert der Aufruf `$\vektor y$` als Ergebnis $(y_1, \ldots, y_n)$, und `$\vektor \lambda$` bietet $(\lambda_1, \ldots, \lambda_n)$.

Makros dürfen bis zu neun Parameter besitzen. Diese müssen aufsteigend mit `#1`, `#2`, `#3` ... `#9` definiert werden. Beim Aufruf eines Makros muß das TEX-Programm entscheiden, wie viel des nachfolgenden Textes jetzt zu den einzelnen Parametern übernommen werden muß. Wird ein Makro ohne spezielle Trennzeichen definiert, was der Normalfall ist, so übernimmt TEX immer das nächste *token*. Dieses kann aber auch wieder der Aufruf eines Makros sein. So ist im vorangegangenen Beispiel im Aufruf "`\vektor\lambda`" der Befehl "`\lambda`" praktisch wieder ein Makro. Ein Makro

```
\def\test#1#2#3{#1#1#2#2#3#3}
```

das mit `\test abcde` im Eingabetext aufgerufen wird, übernimmt für

`#1`	⟵	a
`#2`	⟵	b
`#3`	⟵	c

Das Ergebnis des Makroaufrufes und des nachfolgenden Textes entspricht also der Eingabe `aabbccde`. Die Zeichen '`de`' gehen als *normale Zeichen* an TEX weiter.

Wird zusätzlich noch ein Befehl

```
\def\abc{Alphabet}
```

definiert, bewirkt der Befehl

```
\test\abc abc
```

eine Parameterbesetzung

`#1`	⟵	`\abc`
`#2`	⟵	a
`#3`	⟵	b

Anschließend wird in der weiteren Abarbeitung weiter expandiert:

`#1` ⟵ `\abc` ⟵ Alphabet

Also wird die gesamte Eingabe zu der Zeichenfolge:

```
AlphabetAlphabetaabbc
```

Will man mehr als ein 'Zeichen' oder einen Befehl in *einem* Parameter übergeben, so ist zu klammern:
Der Aufruf im Text

```
\test a{bcde}{fg}hij
```

ermittelt die drei Parameter zu

`#1` ⟵ a
`#2` ⟵ bcde
`#3` ⟵ fg

Die Zeichen '`hij`' sind weiterer Text der normalen Eingabe und werden *nicht* durch das Makro `\test` behandelt.

Nun kann man bei der Definition eines Makros auch selbst angeben, durch welche Zeichen die einzelnen Parameter *getrennt* werden sollen. Die Definition

```
\def\TEST #1 #2 #3 {#1#1#2#2#3#3}
```

für `\TEST` unterscheidet sich von `\test` darin, daß die einzelnen Parameter durch *Leerzeichen* getrennt werden müssen. Der `\test` entsprechende Aufruf lautet

```
\TEST a bcde fg hij
```

Die Einsetzung der Parameter ist die gleiche wie im letzten Beispiel. Bei der Definition des *trennenden* Textes braucht man sich aber nicht auf Leerzeichen zu beschränken. Durch

```
\def\restfett#1.{{\bf#1.}}
```

wird der nachfolgende Text bis zum nächsten Punkt als Parameter übernommen. Allerdings ist noch zu bemerken, daß der Trenntext außerhalb von Klammerstrukturen stehen muß.
Bei der Benutzung in folgendem Text

```
\restfett Ein Text {soll ... hier} folgen. Nun denn
```

wird für den ersten (und einzigen) Parameter eingesetzt:

`#1` ⟵ `Ein Text {soll ... hier} folgen`

Der Trenntext (".") gehört *nicht* zum Parameter!
Auch solche Strukturen sind möglich:

```
\def\meintest #1ABC#2.#3${...}
```

Die Parameter bestehen dann aus dem Text bis zum ersten Auftreten von '`ABC`'. Der zweite Parameter besteht aus den Zeichen *zwischen* '`ABC`' und dem nächsten Punkt. Der dritte aus den Zeichen zwischen Punkt und dem nächsten Dollar.

7.3 Makros innerhalb von Makros

Innerhalb von Makros können selbst wieder neue Makros definiert werden. Ob diese dann nach der Abarbeitung des äußeren Makros noch bekannt sind, hängt von der Blockstruktur ab. Beliebt ist es, lokal Hilfsmakros zu definieren, die dann hinterher nicht mehr vorhanden sind.

Bei der Definition von Makros innerhalb anderer ist auf die Verwendung von `#` zu achten. Es ist dann für innere Makroparameter eines neuen internen Makros ein zusätzliches '`#`' zu setzen.

```
\def\MeineMatrix#1{{\def\vektor##1{#1_##1,\ldots,#1_n}
                   $$\pmatrix{
                         \vektor1\cr
                         \vektor2\cr
                         \vektor3\cr
                         \vektor4\cr}$$}}
```

`\MeineMatrix` besitzt das Untermakro `\vektor#1{ ... }`. Ein Aufruf `\MeineMatrix a` liefert

$$\begin{pmatrix} a_1,\ldots,a_n \\ a_2,\ldots,a_n \\ a_3,\ldots,a_n \\ a_4,\ldots,a_n \end{pmatrix}$$

bzw. `\MeineMatrix{\sin\alpha}` liefert

$$\begin{pmatrix} \sin\alpha_1,\ldots,\sin\alpha_n \\ \sin\alpha_2,\ldots,\sin\alpha_n \\ \sin\alpha_3,\ldots,\sin\alpha_n \\ \sin\alpha_4,\ldots,\sin\alpha_n \end{pmatrix}$$

Soll jedoch umgekehrt zum bisherigen Verhalten ein Makro, das innerhalb einer inneren Klammerstruktur definiert wird, auch nach Verlassen derselben bekannt sein, muß man ein "`\global`" davor setzen.

```
{{{{\global\def\ABC{abcdefghijklmnopqrstuvwxyz}}}}
```

`\ABC` ist auch nach der Abarbeitung des Klammergebirges bekannt. Das gleiche gilt für

```
\def\initABC{{\global\def\ABC{ABCDEFGHIJKLMNOPQRSTUVWXYZ}}
```

Nach dem Aufruf `\initABC` ist das Makro `\ABC` bekannt.

Will man sich die *aktuelle* Bedeutung irgendeines Befehls oder Makros speichern oder einen Befehl umbenennen, hilft der `\let`-Befehl weiter. Durch

```
\let\INITABC=\initABC
\def\initABC{{\global\def\ABC{abcdefghijklmnopqrstuvwxyz}}
```

stehen anschließend zwei Befehle zur Verfügung: `\INITABC` mit der *alten* Bedeutung von `\initABC` und ein neues `\initABC`. Der Befehl "`\let`" *kopiert* also die *aktuelle* Bedeutung eines Befehls auf einen neuen Befehl.

7.4 Expandierung von Makrobefehlen

Bei der Eingabe eines Makros werden die im Definitionsteil stehenden Befehle nur gespeichert. Sie werden bis auf ganz wenige Ausnahmen† noch nicht interpretiert, sondern

† Eine solche Ausnahme ist der Befehl `\newwrite`.

nur gespeichert. Es wird auch noch nicht geprüft, ob diese Befehle überhaupt existieren. Erst zur *Ausführungszeit* werden die Eingabebefehle der Reihe nach ausgewertet. Dies bedeutet insbesondere, daß diese auch mit ihrer zu diesem Zeitpunkt gerade gültigen Bedeutung verwendet werden.

```
\def\ueberschrift{\bigskip\centerline{\bf \titel}\bigskip}
```

möge einen Befehl "`\ueberschrift`" definieren, der den Inhalt von "`\titel`" in fetter Schrift mit Abstand nach oben und unten ausgibt. Durch ein späteres Setzen in "`\def\titel{Computer Architektur}`" wird die aktuelle Bedeutung von "`\titel`" gerade erst festgelegt und beim folgenden Aufruf in "`\ueberschrift`" verwendet. Es kann einem dabei allerdings geschehen, daß ein wichtiger bereits vorhandener TEX-Befehl unabsichtlich überdefiniert wird und damit bestimmte Makros des plain-TEX nicht mehr funktionieren. In der Praxis hat sich der Befehl "`\big`" als Hauptkandidat für diesen Effekt herausgestellt. Dieser Befehl wird im Mathematiksatz verwendet. Viele Anwender neigen dazu, unter diesem Namen eine Schrift zu definieren.

Allerdings bietet der Befehl "`\edef`" *expanded definition* nun gerade die Möglichkeit, den Inhalt einer Makrodefinition schon bei der Definition expandieren zu lassen. Damit wird die aktuelle Bedeutung zum Zeitpunkt der Definition ausgewertet und als Definitionstext eingetragen. Jedoch müssen dann alle zu expandierenden Makros zu diesem Zeitpunkt auch bekannt sein.

```
\def\text{ALT}
\edef\etest{\text}
\def\test{\text}
\def\text{NEU}
```

Danach werden die Aufrufe wie folgt ausgewertet:

`\text`	$\longrightarrow$	NEU
`\test`	$\longrightarrow$	NEU
`\etest`	$\longrightarrow$	ALT

Nun ist es aber bisweilen so, daß ein Teil der Befehle zwar sofort, der andere aber erst später expandiert werden soll. Dazu gibt es den Steuerbefehl "`\noexpand`", der die Interpretation des folgenden Befehls unterdrückt. Die Wirkungsweise sei an dem etwas abgewandelten letzten Beispiel dargestellt. Dort soll zweimal die Anweisung "`\text`" aufgerufen werden, einmal davon mit "`\noexpand`":

```
\def\text{ALT}
\edef\etest{\text---\noexpand\text}
\def\text{NEU}
```

Dies führt zur Ausgabe

`\text`	$\longrightarrow$	NEU
`\etest`	$\longrightarrow$	ALT—NEU

Wird der Befehl "`\noexpand`" vorangestellt, wird die Expandierung unterdrückt. Es wird auch nicht mehr geprüft, ob der folgende Befehl zu diesem Zeitpunkt schon existiert.

Auch dem Befehl "`\edef`" kann ein "`\global`" vorangestellt werden, um diese Definition als *global* zu kennzeichen. Für die Befehlsfolge "`\global\edef`" gibt es den abkürzenden Befehl "`\xdef`"; die Befehlsfolge "`\global\def`" läßt sich mit "`\gdef`" kürzer schreiben.

Noch zwei weitere Makrodefinitionen charakterisierende Befehle sind vorhanden:

`\outer` Damit wird ein Makro mit der Eigenschaft versehen, daß es nur auf dem äußeren Eingabeniveau aufgerufen werden darf. Damit werden Aufrufe in Makros oder als Makroparameter oder in Boxen als Fehler interpretiert. Ein typisches Beispiel für einen solchen Befehl ist das "`\bye`"-Kommando. Es ist sinnvoll, daß dieses nun nicht gerade im Inneren einer verschachtelten Struktur erwartet wird.

`\long` Beim Aufruf eines Makros sind im Normalfall nur *kurze* Parameter erlaubt. So werden also keine ganzen Absätze akzeptiert, es sei denn, bei der Definition wurde diese mit "`\long\def..`" angegeben. Dies führt im anderen Fall zur Fehlermeldung "`runaway arguments`". Damit wird also gewissermaßen eine Notbremse gegen falsch geklammerte Eingaben gezogen.

7.5 Abfragen

Die bisherige Abarbeitung eines Makros war recht statisch: Stets wurde bei gleichem Aufruf auch ein gleiches Ergebnis erzeugt. Häufig möchte man jedoch abhängig von äußeren Bedingungen unterschiedliche Verhaltensweisen erzeugen. Ein Beispiel ist eine Definition der Seitenunterschrift, die für gerade und ungerade Seitennummern verschiedene Ergebnisse erzeugt. Dazu gibt es Abfragebefehle mit der Syntax:

`\if`<*Bedingung*><*wahr-Teil*>`\else`<*falsch-Teil*>`\fi`

Falls die Bedingung erfüllt ist, werden nur die Befehle des *wahr-Teils* abgearbeitet, sonst nur die Befehle des *falsch-Teils*. Wichtig ist dabei auch, daß die Makros, die in einem *true*-Zweig oder *false*-Zweig stehen, nur dann expandiert werden, wenn dieser auch durchlaufen wird.

Fast selbstverständlich ist eigentlich der Hinweis, daß zu jedem `\if...` Befehl auch ein passendes `\fi` gehören muß, sonst wird nicht nur das TEX irritiert, sondern auch der Anwender durch die folgenden Fehlermeldungen.

Es gibt eine ganze Reihe von Bedingungen, die verwendet werden können. Die Folge "`\if`<*Bedingung*>" ist jeweils ein TEX-Befehl. Zwei verschiedene Abfragetypen sind vorhanden: der Größenvergleich und die Abfrage eines bestimmten Zustandes.

Größenvergleiche für Zahlen und Längen

Die folgenden drei Abfragen prüfen, ob die linke Zahl beziehungsweise Dimension kleiner, gleich oder größer als die rechte ist.

Zahlenvergleiche:

`\ifnum` *zahl*$_1$ `<` *zahl*$_2$
`\ifnum` *zahl*$_1$ `=` *zahl*$_2$
`\ifnum` *zahl*$_1$ `>` *zahl*$_2$

Längenvergleiche:

`\ifdim` *dimension*$_1$ `<` *dimension*$_2$
`\ifdim` *dimension*$_1$ `=` *dimension*$_2$
`\ifdim` *dimension*$_1$ `>` *dimension*$_2$

Aufrufbeispiele:

```
\ifnum\pageno=1         ... \else  ... \fi
\ifnum 17>\count0       ... \else  ... \fi
\ifdim\leftskip>\hsize  ... \else  ... \fi
\ifdim\ht0>1cm          ... \else  ... \fi
```

Ein Anwendungsbeispiel: Die Seitennumerierung soll für die erste Seite nicht ausgegeben werden.

```
\footline={\ifnum\pageno=1\hss\else\hss\folio\hss\fi}
```

Vergleiche

Die zweite Gruppe enthält Zustands- und Vergleichsabfragen, mit denen insbesondere auf Identität von Objekten geprüft werden kann. Einige dieser Befehle wird man — wenn überhaupt — nur sehr selten benutzen.

`\ifodd` prüft, ob die nachfolgende Zahlangabe *ungerade* ist.

`\if` prüft, ob die nächsten 2 Zeichen übereinstimmen.
Vorsicht! Dabei werden evtl. folgende Makros expandiert.

`\ifcat` testet, ob die Kategorie-Codes der beiden folgenden Zeichen übereinstimmen.

`\ifx` prüft **ohne volle** Expandierung, ob die beiden folgenden Sequenzen übereinstimmen. Damit können Makrodefinitionen verglichen werden. (Siehe unten das folgende Beispiel!)

Eine beliebte Anwendung ist die oben erwähnte unterschiedliche Seitennumerierung. Die folgende Anweisungsfolge bewirkt bei ungeraden Seitenzahlen eine rechtsbündige Seitennummer und bei geraden Seitenzahlen eine linksbündige Ausgabe:

```
\footline={\rm\ifodd\pageno\hss\folio\else\folio\hss\fi}
```

Wechselnde Seitenüberschriften realisiert man am sinnvollsten durch

```
\def\linkertitel{\it\hfill Autor \hfill}
\def\rechtertitel{\it\hfill Sein Titel\hfill}
\headline={\rm\ifodd\pageno\rechtertitel\else\linkertitel\fi}
```

Auf den Seiten mit ungerader Seitenzahl steht jeweils *'Sein Titel'* als zentrierte Seitenüberschrift und auf den anderen Seiten entsprechend *'Autor'*.

Eine weitere Anwendung ist die Prüfung, ob ein Makroparameter beim Aufruf unbesetzt ist. Dazu wird zunächst ein *leeres* Makro `\empty` definiert, gegen das geprüft wird. (Die Definition von "`\empty`" kann auch entfallen, sie ist in plain-TEX sowieso vorhanden.) Innerhalb der Definition muß noch ein zweites Vergleichsmakro — hier `\test` — als Gegenstück definiert werden. Dann können diese beiden mit Hilfe von `\ifx` miteinander verglichen werden.

```
\def\empty{}                % Vergleichsmakro mit leeren Inhalt
\def\meinMakro#1#2#3
     {\def\test{#1}         % Makro, dessen Inhalt gleich #1 ist
      \ifx\test\empty  ... % Anweisungen, falls #1 unbesetzt ist
         \else         ... % Anweisungen, falls #1 nicht unbesetzt ist
         \fi}
```

Wird zum Beispiel "`\meinMakro{}{ABC}{DEF}`" aufgerufen, expandiert '`#1`' zum (leeren) Innern von "`\def\test{}`". Dies stimmt aber mit dem Definitionsteil von `\empty` überein. Beim Aufruf mit "`\meinMakro{eins}{zwei}{drei}`" wird dann in der Abarbeitung das lokale Prüfmakro zu "`\def\test{eins}`" gebildet, daher ist beim Vergleich "`{}`" und "`{eins}`" verschieden.

Abfragen des Satzmodus

Es folgen noch einige Befehle, die prüfen, in welchem der verschiedenen TEX-Arbeitsmodi gerade gesetzt wird. So kann innerhalb eines Makros entschieden werden, ob es innerhalb einer mathematischen Formel oder im Fließtext aufgerufen wurde.

`\ifvmode` prüft auf *vertical mode* oder *internal vertical mode*

`\ifhmode` prüft auf *horizontal mode* oder *restricted horizontal mode*

`\ifmmode` prüft auf *mathematical mode*

`\ifinner` prüft auf *internal mode*

Dies ist im Innern einer "`\vbox`" im *internal vertical mode* und im Innern einer "`\hbox`" als *restricted horizontal mode* gesetzt.

Mit Hilfe von "`\ifvmode`" und "`\ifhmode`" kann vollständig unterschieden werden, in welchem der Arbeitsmodi man sich befindet.

Ein Beispiel: Der Befehl "`\stars`" soll unabhängig von Text- und Mathematikmodus die drei Sterne "⋆⋆⋆" ausgeben. Dies leistet die folgende Definition:

```
\def\stars{\ifmmode
               {\star}{\star}{\star}
           \else${\star}{\star}{\star}$%
           \fi}
```

Die Befehle `\centerline` und die Gegenstücke `\leftline` beziehungsweise `\rightline` haben die unangenehme Eigenschaft, daß sie einen Absatz nicht beenden, falls sie versehentlich innerhalb eines Absatzes aufgerufen werden. Dies endet dann in einer

sehr unschönen Zeile: Ein neuer Befehl `\Centerline`

```
\def\Centerline{\ihmode
                  \ifinner
                  \else
                     \par
                  \fi
                \fi
                \centerline}
```

prüft, ob er innerhalb eines Absatzes aufgerufen wurde, und beendet gegebenenfalls den Absatz bevor er quasi durchstartend den alten `\centerline` Befehl ausgibt.

Abfragen der Box-Register

Weitere Abfragen im Zusammenhang mit Boxregistern sind die folgenden Befehle. Diese sind jedoch nur im Zusammenspiel mit "`\setbox`" zu verwenden. Dieser Befehl besetzt ein Boxregister mit Inhalt. Die Beschreibung dazu folgt im nächsten Kapitel.

`\ifhbox` prüft, ob die folgende angegebene Box eine "`\hbox`" enthält.
`\ifvbox` prüft, ob die folgende angegebene Box eine "`\vbox`" enthält.
`\ifvoid` prüft, ob die folgende angegebene Box leer ist.

Dazu seien dennoch vorab einige Beispiele angegeben. Nach den Besetzungen

```
\setbox0=\hbox{Das ist ein Wort!}      % Besetzung
\setbox1=\vbox{\hrule\vskip1cm\hrule}  %
\setbox2=\hbox{}                       %
\box3                                  % Ausgabe und Leerung
```

liefert etwa `\ifhbox1` den Wert "*false*". Insgesamt erhält man:

	$n=0$	$n=1$	$n=2$	$n=3$
`\ifhbox` n	true	false	true	false
`\ifvbox` n	false	true	false	false
`\ifvoid` n	false	false	false	true

7.6 Eigene if-Befehle

Zur Abfrage und Erzeugung eigener *if*-Befehle sind drei Befehle vorhanden, die diese Konstruktionen unterstützen. Zunächst sind dies die beiden Befehle "`\iftrue`" und "`\iffalse`". Diese liefern beim Aufruf immer den Wert *true* beziehungsweise *false*. Dies scheint für eine *Abfrage* nun zunächst gar nicht sehr sinnvoll. Durch eine Konstruktion mittels des Befehls "`\newif`" werden diese Befehle aber sehr brauchbar.
Nach "`\newif\ifvorwort`" werden drei Makros erzeugt. Dies sind:

`\ifvorwort` zur Abfrage
`\vorworttrue` zum Setzen von *true*; genau betrachtet wird `\ifvorwort` danach zu einem `\iftrue`.
`\vorwortfalse` zum Setzen von *false*; genau betrachtet wird `\ifvorwort` danach zu einem `\iffalse`.

Nach der Deklaration ist der Zustand *false* voreingestellt. Damit kann in einer Anwendung sehr schön und vor allen Dingen lesbar mittels "`\ifvorwort`" eine saubere Abfrage formuliert werden.

Die Befehle "`\ifvorwort ... \else ... \fi`" können dann in der gleichen syntaktischen Form wie die normalen `\if`-Anweisungen verwendet werden.

Durch die neuen Befehle `\ifvorworttrue` und `\ifvorwortfalse` werden übrigens die Anweisungen `\let\ifvorwort=\iftrue` und `\let\ifvorwort=\iffalse` ausgeführt.

Beim Aufruf des Befehls "`\newif`" muß der folgende Befehl mit der Zeichenfolge "`\if`" beginnen, sonst erfolgt eine Fehlermeldung. Sehr empfehlenswert ist ein Befehl wie `\ifdebug` oder `\iftest` mit dem Testausgaben, die während der Entwicklungsphase von komplizierten Makros auftreten, geeignet ein- oder ausgeschaltet werden können.

7.7 Testen der Makros

Da wohl niemand im ersten Anlauf völlig fehlerfreie Makros schreibt, sondern sich die Makroentwicklung häufig als ein Versuch-und-Irrtum-Prozeß herausstellt, sind einige Hinweise zum Testen angebracht.

Protokollierung der Makrobedeutung

Der Befehl `\meaning` gibt die Bedeutung eines Makros aus. Mittels `\message` kann man sich einen beliebigen Text ins Protokoll und auf den Bildschirm schreiben lassen. Dies ist sehr gut für Testausgaben bei der Entwicklung von komplizierten Makros zu verwenden.
Nach einer Definition

```
\def\TestObDefiniert#1{%
        \ifx#1\GanzBestimmtUndefiniert
           \message{undefiniert}%
        \else
           \message{definiert}%
        \fi}
```

erzeugt der Befehl `\message{\meaning\TestObDefiniert}` die folgende Ausgabe

```
macro:#1->\ifx #1\GanzBestimmtUndefiniert \message {definiert}
\else \message {undefiniert}\fi
```

Der Befehl `\meaning` gibt die Definition eines Makros als fortlaufende Zeichenfolge aus, ohne diese an Positionen wie `\if...` oder `\else` in einzelne Zeilen aufzuteilen.

Der oben definierte Befehl `\TestObDefiniert` prüft `\ifx` durch einen Vergleich gegen ein hoffentlich nicht vorhandenes Makro (`\GanzBestimmtUndefiniert`), ob ein Befehl schon existiert und gibt eine entsprechende Meldung aus.

Wenn man möchte, kann man sogar den `\def` Befehl dahin gehend ändern, daß eine Warnung erzeugt wird, falls der neu zu erzeugende Befehl schon vorhanden ist.

Dies geschieht durch folgende Befehlsfolge:*

```
\let\AltesDef=\def
\AltesDef\def#1{\ifx#1\GanzBestimmtUndefiniert
        \else
        \message{\string#1 ist schon als Makro definiert}%
        \fi\AltesDef#1}
```

Zunächst wird mittels `\let` das alte `\def` *kopiert*, denn dieser Befehl wird noch gebraucht. Anschließend wird ein neuer `\def`-Befehl erzeugt, der prüft, ob der Name schon vergeben ist. Wird übrigens der Befehl `\message` durch `\errmessage` ersetzt, so hält das Programm sogar an und verlangt eine vorrangige Eingabe.

Protokollierungen

Im Vorgriff auf den Abschnitt 10.3 sei hier nur darauf hingewiesen, daß mittels der `\tracing`-Befehle sehr umfangreiche Protokollierungen der ausgeführten Anweisungen aktiviert werden können.

7.8 Trick-Makros oder Makros für Fortgeschrittene

Die folgenden Befehle sind nur für eine sehr extreme Nutzung des TEX sinnvoll. Dieser Abschnitt kann zunächst auch ausgelassen werden!

Gruppenstrukturen und Makros

Bisweilen ist es notwendig, in einem Makro eine Gruppe zu eröffnen, die dann durch ein anderes Makro geschlossen werden soll. Bei der Definition eines Makros müssen aber die öffnenden Klammern "`{`" und die schließenden "`}`" genau paarig zu einander sein. Hier helfen die Befehle `\bgroup` und `\egroup` beziehungsweise `\begingroup` und `\endgroup`. Durch `\bgroup` und `\begingroup` wird wie durch `{` eine neue Gruppe eröffnet. Es sind auch Konstruktionen möglich, die eine Box eröffnen "`\hbox\bgroup`". Eine Gruppe, die durch `\bgroup` oder auch wie gewohnt durch `{` begonnen wurde, darf entweder durch `}` oder `\egroup` beendet werden. Dagegen müssen `\begingroup` und `\endgroup` genau als Paare auftreten, sonst erfolgt eine Fehlermeldung. Dies hat zum Testen die angenehme Wirkung, daß man damit prüfen kann, ob die Gruppenklammern auch genau so aufgehen, wie man sich das gedacht hatte.

Änderung der Expandierungsreihenfolge

Die Makrotechnik des TEX-Programms ist recht kompliziert. Es gibt eine Reihe von zusätzlichen Befehlen, die komplizierte Steuerungsmöglichkeiten bieten.

`\afterassignment` Der auf `\afterassignment` folgende Befehl wird *nicht sofort* ausgeführt, sondern gespeichert und erst *nach der nächsten Zuweisung* ausgeführt.

---

* Dieses Makro funktioniert übrigens nicht für den Test gegen den `\relax` Befehl, da dieser ebenso wie ein undefinierter Befehl, hier `\GanzBestimmtUndefiniert` durch `\ifx` zu nichts expandiert.

Als Beispiel diene ein Makro, das alle kleinen Vokale in einem Wort in einer anderen Schrift druckt. Der hier verwendete Befehl `\xyz` wählt dabei einen neuen Font an und unterstreicht das Zeichen.

```
\def\xyz{\bf\underbar}
\def\vokale#1{\dovokale#1\endlist}
\def\endlist{\endlist}
\def\dovokale{\afterassignment\dobf\let\next= }
\def\dobf{%
   \ifx\next\endlist\let\next\relax
   \else
      \if\next a{\xyz a}%
      \else\if\next e{\xyz e}%
           \else\if\next i{\xyz i}%
                \else\if\next o{\xyz o}%
                     \else\if\next u{\xyz u}%
                          \else\next
                          \fi
                     \fi
                \fi
           \fi
      \fi
      \let\next\dovokale
   \fi\next}
```

Beispiel: `\vokale{fundamental}` liefert "f**u**nd**a**m**e**nt**a**l"
Die Abarbeitung ist etwas kompliziert. Zunächst wird eine *Bremse* benötigt, um das Ende der Buchstabenliste festzustellen. Dazu dient das Makro `\def\endlist{\endlist}`. Für den Normalgebrauch ist es unzureichend, da es nur eine Endlosschleife liefern würde. Hier wird mittels `\ifx` der Inhalt von `\next` und `\endlist` verglichen. Die Prozedur `\vokale` ruft die parameterlose Prozedur `\dovokale` auf. Diese wendet nun den Befehl `\afterassignment` an. Der Aufruf der weiteren Prozedur `\dobf` wird vorgespeichert. Durch `\let\next=` wird eine Zuweisung aktiviert und zwar auf die folgenden Zeichen. In unserem Beispiel wäre dies als erstes das 'f'. Durch die Vorspeicherung des `\dobf`-Aufrufes wird anschließend der Befehl `\dobf` aufgerufen mit der Besetzung "`\next=f`". Dort wird mit einer `\if`-Kaskade geprüft, ob ein Vokal vorliegt, und dieser in bold gesetzt. Die äußeren Abfragen dienen nur dazu, festzustellen, ob das Ende des `\vokale`-Parameters erreicht ist. Falls nicht, wird `\dovokale` noch einmal aufgerufen.

In einem zweiten Beispiel soll ein Befehl zum A u s s p e r r e n von Wörtern dargestellt werden. Hier liegt ja das gleiche Problem vor, daß unbekannt ist, wie viele Buchstaben folgen. Nach der folgenden Befehlsfolge liefert "`\sperr{Aussperren}`": A u s s p e r r e n.
Übrigens: Dieses Makro funktioniert nicht mit Umlauten.

```
\def\sperr#1{\SperrRest#1\endlist}
\def\endlist{\endlist}
\def\SperrRest{\afterassignment\SperrZeichen\let\next= }
\def\SperrZeichen{\ifx\next\endlist \let\next\relax
                                   \kern-0.25em
                  \else            \next \kern0.25em
                                   \let\next\SperrRest
                  \fi
                  \next}
```

Hiermit wird gleichzeitig eine Schleifentechnik zur Abarbeitung unterschiedlich langer Parametertexte demonstriert. Es sei hier noch einmal die Abarbeitungsfolge dargestellt:

1. `\sperr{Aussperren}`
2. `\SperrRest Aussperren \endlist`
3. `\afterassignment\Sperrzeichen \let\next=Aussperren\endlist`
4. Der Befehl "`\Sperrzeichen`" wird gespeichert. Eingabezustand: `\let\next=Aussperren\endlist`
5. Ausführung der Zuweisung "`\let\next=A`"
6. Der gespeicherte Befehl wird zurückgeholt: Eingabezustand: `\Sperrzeichen ussperren\endlist`
7. Das Makro `\Sperrzeichen` läuft ab. Am Ende wird der Befehl "`\SperrRest`" abgesetzt.
8. Eingabezustand: `\SperrRest ussperren \endlist` Dies ist der gleiche Zustand wie bei 3.

Soll vor und nach dem Wort ebenfalls ausgesperrt werden, so sind die Makros wie folgt zu ändern:

```
\def\sperr#1{\kern0.25em\SperrRest#1\endlist}
```

Im Makro `\Sperrzeichen` ist der Befehl "`\kern-0.25em`" zu streichen.

`\expandafter` Mit diesem Befehl wird die Reihenfolge, in der die einzelnen TEX-Befehle *expandiert* werden, verändert. *Expandieren* geschieht zum Beispiel bei Makros. Der nachfolgende Befehl wird erst nach dem übernächsten Befehl expandiert.

```
\def\nextbf#1{{\bf #1}}
\def\meintext{Das ist ein Beispielsatz!}
```

Das Makro `\nextbf` setzt das nächste Zeichen in der Schriftart bold. `\meintext` ist eine einfache Textabkürzung.

Beispiel:
`\expandafter\nextbf\meintext`
liefert
Das ist ein Beispielsatz!

Beispiel:
`\nextbf\meintext`
liefert
Das ist ein Beispielsatz!

`\futurelet` Dieser Befehl dient der Inspektion der folgenden Befehle. Am besten läßt sich seine Wirkung durch die folgende Syntaxangabe beschreiben:

`\futurelet\`*next*`\`*test*`\`*weiter*

Damit wird dem Befehl `\`*next* der Inhalt von `\`*weiter* zugewiesen. Der Befehl `\`*weiter* bleibt aber in der Eingabe stehen. Das Makro `\`*test* wird aufgerufen und kann die folgende Information schon im Inhalt von `\`*next* prüfen.
Die Wirkung entspricht der Befehlsfolge:

`\let\`*next*`\`*weiter*`\`*test*`\`*weiter*

Dieser Befehl sei an folgendem Problem demonstriert: Es soll ein Befehl erzeugt werden, der sich wie ein Standardbefehl im Mathematiksatz verhält, daß heißt, er soll Parameter besitzen, die durch "`_`" (Unterstrich) und "`^`" eingeleitet werden. Das Problem dabei ist, daß diese aber auch fehlen können. Gesucht ist ein eigener Befehl "`\myop`", der in den Varianten "`\myop`", "`\myop_{..}`", "`\myop^{..}`" und "`\myop_{..}^{..}`" aufgerufen werden darf. Dieses Problem läßt sich nur dadurch lösen, daß man einen Blick auf die nachfolgende Information wirft.

```
\def\myop{\def\myopUP{\infty}
          \def\myopDOWN{\infty}
          \futurelet\next\myopGO}
\let\sb=_  % Standard in plain-tex
\let\sp=^  %
\def\myopGO{\ifx\next\sb\let\next\doDOWN
            \else
               \ifx\next\sp\let\next\doUP
               \else\let\next\UPandDOWN
               \fi
            \fi\next}
\def\doDOWN_#1{\def\myopDOWN{#1}\futurelet\next\myopGO}
\def\doUP^#1{\def\myopUP{#1}\futurelet\next\myopGO}
\def\UPandDOWN{%
    {\vphantom{\big\Vert}%
        _{\myopDOWN}\big\Vert
        ^{\myopUP}}}
```

Beispiel:

`$$ \myop $$` *liefert* $\myop$

`$$ \myop_\alpha $$` *liefert* $\myop_\alpha$

`$$ \myop^\beta $$` *liefert* $\myop^\beta$

`$$ \myop_{aa}^{bb} $$` *liefert* $\myop_{aa}^{bb}$

Beispiel für Makros mit optionalen Parametern

In dem folgenden — zugegeben komplexen Beispiel — wird demonstriert, welche Fähigkeit TEX besitzt, um vom Standpunkt des späteren Anwenders auch syntaktische Erweiterungen eines Standardkommandos zu erreichen.

Es wird ein Befehl `\optional` definiert, der einer normalen `\def`-Anweisung vorangestellt werden kann, und damit in ähnlicher Weise wie `\long` oder `\global` verwendet wird. Die notwendigen Makros und die Erläuterungen dazu sind auf den beiden folgenden Seiten einander gegenüber gestellt.
Als Ergebnis für ein neues Kommando "`\`*test*" erhält man

`\optional\def\`*test*`[`*Vorbelegung*`]#2...{...}`

Das Kommando `\test` kann dann so verwendet werden, als sei es definiert durch

```
\def\test[#1]#2...{...}
```

aber die Information für den ersten Parameter `#1` ist vorbelegt. Als Beispiel sei eine Abwandlung des `\footnote` Befehls durch

```
\optional\def\FootNote[$^\ast$]{\footnote{#1}}
```

gegeben, die den neuen Befehl `\FootNote` so definiert, daß er als Anweisung `\FootNote{Fußnotentext}` oder auch als `\FootNote[$^1$]{Fußnotentext}` verwendet werden kann. Im ersten Fall wird für die Markierung "`$^\ast$`" verwendet, im zweiten Fall wird diese Vorbelegung überschrieben durch "`$^1$`". Im Beispiel

```
\optional\def\test[ABCD]#2{Erster Wert: #1, zweiter Wert: #2.}
```

erhalten wir bei der Eingabe

`\test{1234}` für die Parameter die Werte `#1` ⟵ ABCD, `#2` ⟵ 1234

und als Ergebnis "Erster Wert: ABCD, zweiter Wert: 1234"

`\test[1]{2}` liefert für die Parameter `#1` ⟵ 1, `#2` ⟵ 2

und für als Ergebnis: "Erster Wert: 1, zweiter Wert: 2"

Die Konstruktion des Makros `\optional` ist recht kompliziert. Sie wird in mehreren Schritten vollzogen. Abhängig vom Namen des Makros, das neu definiert werden soll, werden einige zusätzliche Hilfsmakros erzeugt, die dadurch gegenüber dem Anwender versteckt werden, daß sie das Zeichen "`@`" im Namen enthalten, welches zu diesem Zeitpunkt wie ein Buchstabe behandelt wird. Diese Technik der Umstellung des `\catcode` Wertes wird auch häufig in plain-TEX verwendet.

Angenommen, der neue Befehl soll "`\test`" heißen, so werden abhängig von diesem Namen folgende zusätzliche Befehle erzeugt:

Makrodefinition

```
\catcode`\@=11                  % Einige Befehle werden versteckt.
\def\optional#1#2[#3]{%
   \escapechar=-1

   \if\def#1%

      \edef#2{\futurelet\noexpand\next
      \csname\string#2@@body\endcsname}%

      \expandafter\edef\csname\string#2@@body\endcsname{%
           \noexpand\if[\noexpand\next
               \def\noexpand\next
                  {\csname\string#2@@do\endcsname}%
           \noexpand\else
               \def\noexpand\next{%
                 \expandafter\noexpand
                     \csname\string#2@@do\endcsname
                        [\expandafter\noexpand
                         \csname\string#2@@default\endcsname]}%
           \noexpand\fi
           \noexpand\next}%

      \expandafter\noexpand\expandafter
             \def\csname\string#2@@default\endcsname{#3}%

      \edef\optional@continue{\expandafter\noexpand
          \expandafter\def\csname\string#2@@do\endcsname[####1]}%

      \escapechar="5C\relax

      \let\next=\optional@continue

   \else
      \escapechar="5C
      \errmessage{\string\def fehlt nach \string\optional}%
      \let\next=\relax

   \fi
   \next}%
\catcode`\@=12
```

Erläuterungen für einen Aufruf: `\optional\def\test`

- Durch diesen kleinen Trick wird das Zeichen "`@`" behandelt wie ein gewöhnlicher Buchstabe. Der erste Parameter von `\optional` wird "`\def`", der zweite Parameter den Namen des neuen Makros, zum Beispiel "`\test`", enthalten. Auf "`#3`" wird schließlich die Vorbelegung, mit eckigen Klammern umschlossen, erwartet.

- Zuerst wird geprüft, ob auch der Befehl "`\def`" folgt.

- Der Befehl *"\test"* wird so definiert, daß er über `\futurelet` das nächste *token* übernimmt, um zu prüfen, ob eventuell ein "`[`"-Zeichen folgt, das die Vorbesetzung überschreibt: `\edef\`*test*`{\futurelet\next\`*test*`@@body}`

- Das prüfende Makro `\test@@body` hat dann die Form

```
\def\test@@body{\if[\next
                    \def\next{\test@@do}
                \else
                    \def\next{\test@@do[\test@@default]}
                \fi
                \next}
```

  Dies geschieht mittels `\edef`. Dabei haben fast alle Kommando ein `\noexpand` vorangestellt, um eine zu frühe Expandierung zu verhindern mit Ausnahme der Befehlsfolge `\csname\string...\endcsname`. Diese wird benötigt um die zusammengesetzten Befehlsnamen zu erzeugen.

- Das Kommando `\test@@default` mit der Vorbelegung für "`#1`" wird definiert.

- Der Befehl `\test@@do[#1]` mit der originären Befehlsfolge des Zielmakros `\test` wird definiert: Da diese Daten noch nicht gelesen sind und daher in der nachfolgenden Eingabe noch immer anstehen, können sie recht einfach übernommen werden. Diese Arbeit wird durch den Befehl `\optional@continue` erledigt, der als letztes Kommando von `\optional` abläuft.

- Der alte Wert von `\escapechar` wird wieder eingesetzt.

- Der letzte Befehl, der durch `\optional` ausgeführt werden soll, wird gesetzt.

- Erzeugung einer Fehlermeldung, falls nicht der Befehl `\def` auf `\optional` folgt.

- Die Definition des Makros `\optional` with mit der Belegung des `\catcode`-Wertes für "`@`" beendet.

`\test@@default` Dieses Makro enthält die Vorbelegung des ersten Parameters, die bei der Makrodefinition mitgegeben wird.

`\test@@body` Bei einem späteren Aufruf von `\test` prüft dieser Befehl, ob das Zeichen "`[`" folgt, und damit die Vorbelegung überschrieben wird. Lautet der Aufruf tatsächlich `\test[...]...` so wird als nächstes der Befehl `\test@@do` ausgeführt, sonst der Befehl
`\test@@do[\test@default]`

`\test@@do` entspricht logisch einer Definition von
`\def\test[#1]#2{Erster Wert: #1, zweiter Wert: #2.}`
Diese Anweisung enthält also den eigentlichen Makrotext. Sie wird durch den Befehl `\test@@body` aufgerufen.

Einige TEX-Kommandos brauchen vielleicht einige zusätzliche Erläuterungen

`\string` erzeugt als Ergebnis einfach den *Namen* des folgenden Kommandos. `\string` kann auch einfach dazu verwendet werden, TEX-Befehle zu drucken, zum Beispiel `\string\bigskip` druckt
"bigskip
mit vorangestellten doppelten Apostrophen. In der normalen Roman-Schrift ist nämlich an der Zeichenposition für den `\escapechar` gerade dieses Zeichen. Um das korrekte Zeichen zu erhalten, muß in eine passende Schrift gewechselt werden. Bei den Computer Modern Fonts erhält man den gesuchten inversen Schrägstrich im `typewriter` Zeichensatz. `\tt\string\bigskip` erzeugt
`\bigskip`.

`\escapechar` Dies ist ein internes TEX-Register mit dem Codewert, der für die Ausgabe des *escape*-Zeichens verwendet werden soll. Wird der Wert auf -1 gesetzt, so wird die Ausgabe unterdrückt. Normalerweise steht der Wert auf dezimal 92, dies ist gerade der Wert für \ entsprechend der ASCII-Codetabelle.

`\csname` `\endcsname` `\csname` und `\endcsname`, die immer als Paar auftreten umgeben einen — fast — beliebigen Text, der allerdings *keinen* inversen Schrägstrich (\) enthalten darf. Aus diesem Text wird dann unter Voranstellen eines Schrägstrichs ein TEX-Befehl gebildet. Interessanterweise dürfen bei dieser Methode beliebige Zeichen zur Bildung von TEX-Befehlen verwendet werden.

`\errmessage` protokolliert die Information, die im Parameterteil steht, als Fehlermeldung und hält den TEX-Lauf in gleicher Weise an, wie es auch ein normaler Fehler bewirkt hätte. (Das heißt, bei aktivem *error stop mode* wird eine vorrangige Eingabe angefordert.)

Schleifen

Plain-TEX bietet auch eine Methode zur Schleifenkonstruktion, wie sie etwa in normalen Programmiersprachen enthalten ist. Die `\loop` Anweisung genügt der folgenden syntaktischen Form:

```
\loop $\alpha$ \if... $\beta$ \repeat
```

Für α und β sind beliebige Anweisungen erlaubt, jedoch können im α-Teil direkt keine `\if`-Befehle stehen.

Zunächst werden die Anweisungen für α interpretiert, anschließend wird die `\if...`-Bedingung geprüft. Ist die Bedingung nicht erfüllt, so werden alle Anweisungen bis zum `\repeat` übersprungen. Ist die Bedingung erfüllt, so werden auch die Instruktionen des β-Teils ausgeführt und die Schleife beginnt mit den Befehlen für α von neuem.

Dies sei im folgenden Beispiel demonstriert:

```
\def\DREIECK#1{{\def\bull{}%
                \count1=0
                \loop
                   \edef\bull{$\bullet$\bull}
                \ifnum\count1<#1
                   \advance\count1 by 1
                   \centerline{\bull}
                   \vskip-7.7pt
                \repeat
                \vskip 7.7pt\relax}}
```

Der Aufruf `\DREIECK{10}` liefert das Bild:

Ein anderes Beispiel zeigt eine einfache Methode, um die Zeilen eines Absatzes zu numerieren. Einige Befehle, insbesondere `\vsplit` sind im folgenden Kapitel näher erläutert.

```
\long\def\NumberParagraph#1{%
   {\setbox1=\vbox{\advance\hsize by -20pt#1}%
    \vfuzz=10pt                % Warnungen werden damit unterdrückt
    \splittopskip=0pt          % kein \topskop am Anfang der Box 1
    \count1=0                  % Initialisierung der Zeilenzählung
    \par\noindent              % ein neuer Absatz für die Ausgabe
    \def\rebox{%
        \advance\count1 by 1\relax
        \hbox to 20pt{\strut\hfil\number\count1\hfil}%
        \nobreak
        \setbox2=\vsplit 1 to 6.1pt
        \vbox{\unvbox2\unskip}%
        \hskip 0pt plus 0pt\relax}%
    \loop
        \rebox                 % Bearbeitung einer Zeile
        \ifdim \ht1>0pt        % Test, ob noch mehr
    \repeat                    %           Zeilen vorhanden sind
    \par}}
```

Angewendet in

```
\NumberParagraph{\noindent
    In the sense in which architecture is an art,
    typography is an art. That is, both come under
    the head of ‘‘making or doing intentionally with
    skill.’’ Every work of architecture, every work of
    typography, depends for its success upon the clear
    conveyance of intentions, in words and otherwise,
    from one human mind to others: from the man who
    is supposed to know how the finished thing should
    look and function.   \hfill Beatrice Warde}
```

erhalten wir

1 In the sense in which architecture is an art, typography is an art. That is, both
2 come under the head of “making or doing intentionally with skill.” Every work
3 of architecture, every work of typography, depends for its success upon the clear
4 conveyance of intentions, in words and otherwise, from one human mind to others:
5 from the man who is supposed to know how the finished thing should look and
6 function. Beatrice Warde

In dem Makro `\NumberParagraph` wird zunächst die Information vollständig im Box-Register 1 aufgenommen, dabei wird die Zeilenlänge (`\hsize`) etwas verkleinert, um am Anfang jeder Zeile noch Platz für die Numerierung zu erhalten. Dazu wird der Wert `20 pt` verwendet, der gleich der Standardbelegung von `\parindent`, der Absatzeinrückung, ist.

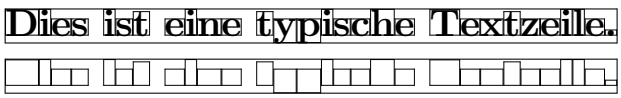
Eine Schleife zerlegt nun Zeile für Zeile das Box-Register 1, wobei ein neuer Absatz gebildet wird, in der jede Zeile mit ihrer Zeilennummer versehen wird. Dazu ist allerdings zu sagen, daß dies *nicht* mit mathematischen Formeln im *display style* funktioniert.

In diesem Beispiel sind schon ein wenig die Anwendungsmöglichkeiten vorweggenommen, die der Umgang mit Boxen in TEX bietet. Dies ist das Thema des nächsten Kapitels.

8 Wie TEX arbeitet

8.1 Kästchen

Die “Arbeitseinheit” des TEX-Programms ist ein rechteckiges Kästchen. Beginnend mit den kleinsten Einheiten, den Buchstaben, wird jedes Zeichen zunächst als ein Kästchen mit gewisser Höhe, Breite und auch Tiefe betrachtet. Die Höhe und Breite sind sofort verständlich, die Tiefe entspricht der Unterlänge, die einige Zeichen — g, j, p, q, y — besitzen. Dies kann man gut mit den Bleilettern des alten Buchsatzes vergleichen. Auch dort wurden die Zeilen nach der ‘Grundlinie’ ausgerichtet. Eine Textfolge “Dies ist eine typische Textzeile.” wird durch TEX als eine Folge von Einzelelementen betrachtet:



Das Trennen einzelner Worte soll unberücksichtigt bleiben. Dann besteht die Eingabezeile aus mehreren größeren Kästchen, den Worten:

TEX faßt Schritt für Schritt die einzelnen Elemente zu größeren zusammen, bis hinterher eine ganze Seite übrigbleibt. Um ein gutes Layout zu erreichen, kann das TEX-Programm die Kästchen in gewissem Umfang *bewegen*. Zum Beispiel wird beim Randausgleich der Platz zwischen den Wörtern vergrößert und verkleinert, um einen gleichmäßigen rechten Rand zu bilden.

8.2 TEX’s interne Arbeitsmodi

Dabei kennt das TEX-Programm bei der Verarbeitung der einzelnen “Kästchen” folgende Arbeitsmodi:

- **vertical mode** und **internal vertical mode**
  Die Kästchen dürfen vertikal — auf und ab — bewegt werden. Sie werden untereinander gesetzt und eventueller *dynamischer Leerraum* (z. B. `\vfill`) bewirkt einen besonderen Ausgleich.
  internal vertical mode ist der Arbeitsmodus, in den TEX durch den `\vbox`-Befehl übergeht. Die nachfolgende Information soll vertikal in der umgebenden Box angeordnet werden. Sie unterliegt dabei nicht dem Seitenumbruch.

Gegenüber dem "normalen" *vertical mode* sind im wesentlichen eine Reihe von Befehlen, wie `\end` oder `\eject` untersagt.
Der normale *vertical mode* ist der Arbeitsmodus, in dem einzelne Zeilen untereinander auf der aktuellen Seite gesetzt werden und der Seitenumbruch vollzogen wird.

- **horizontal mode**
  Dies ist der Arbeitsmodus, in dem Elemente horizontal bewegt werden können. In diesem Modus befindet man sich die meiste Zeit. Das ist der Arbeitsmodus, in dem der Text eines Absatzes umbrochen und zu einzelnen Zeilen geformt wird. Es wird versucht, aus den Wörten einzelne Zeilen zu erzeugen, die möglichst genau die Länge `\hsize` besitzen. Diese bilden dann die Zeilen eines Absatzes. In den *horizontal mode* kommt man durch die Eingabe einfachen Textes oder sonstigen sogenannten *horizontalen Materials*, falls man *vorher* im einem der *vertical modes* war.
  Horizontales Material sind zum Beispiel `\noindent`, `\hskip`, `\indent`, `\vrule`, `\quad` oder `\hfill`, aber auch normale Textzeichen.
- **restricted horizontal mode**
  In diesem Arbeitsmodus können die Elemente *ohne Zeilenumbruch* nur horizontal bewegt werden. Sie werden nebeneinander angeordnet.
  Durch den Befehl `\hbox{ }` wird dieser Arbeitsmodus angewählt.

Es ist jederzeit möglich, in einen anderen Modus zu kommen. Durch verschiedene ineinander geschachtelte `\vbox`- und `\hbox`-Befehle wird regelmäßig gewechselt. Nach dem Verlassen einer Box gilt wieder der Arbeitsmodus, der vorher außerhalb gültig war.

8.3 Box-Manöver

Die Befehle `\vbox` und `\hbox` regeln, wie der Text im *Innern* gesetzt wird. (Es wird nach `\vbox{` bzw. `\hbox{` auch automatisch eine Klammergruppe eröffnet. Damit gehen alle Einstellungen im Innern beim Verlassen dieser Box wieder verloren.) Ist TEX mit der Abarbeitung einer solchen Box fertig, ist diese für TEX nur *ein einziges* großes Kästchen. Dieses Kästchen muß nun gesetzt werden. Die Verwendung und Satzweise dieser Box hängt nun einfach vom Arbeitsmodus der *äußeren* Umgebung ab. Ist dort der Arbeitsmodus *vertical*, wird untereinander gesetzt, ist er *horizontal*, nebeneinander. Dabei wird beim Nebeneinandersetzen die Grundlinie der einzelnen Kästchen als Referenz genommen. Bei einer `\vbox` ist dies die Grundlinie der letzten inneren Box.

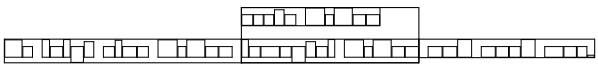
Das folgende Beispiel verdeutlicht das Verhalten: Wir befinden uns dabei im normalen *horizontal mode* — normaler Satz mit Zeilen- und Absatzumbruch durch den Text `\par Es folgt...`. In der `\vbox` wird allerdings im *internal vertical mode* gearbeitet.

```
\par Es folgt eine V-Box
     \vbox{\hbox{erste H-Box}\hbox{‘zweyte’ H-Box}}
     und noch was. \par
```

liefert

erste H-Box
Es folgt eine V-Box ‘zweyte’ H-Box und noch was.

oder mit ‘Rähmchen’:

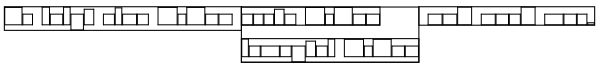


Neben \vbox gibt es noch den Befehl \vtop, da liegt die Grundlinie oben. Das heißt, die Grundlinie wird durch die Grundlinie der ersten Box (Zeile) gebildet. Das letzte Beispiel abgewandelt

```
\par Es folgt eine V-Box
     \vtop{\hbox{erste H-Box}\hbox{‘zweyte’ H-Box}}
     und noch was. \par
```

liefert

Es folgt eine V-Box erste H-Box und noch was.
‘zweyte’ H-Box



Im folgenden Beispiel wird in der \vbox intern durch \noindent in den *horizontal mode* gewechselt. Dadurch wird ein Zeilenumbruch erreicht. Durch \hsize=0.50\hsize wurde die Zeilenlänge lokal für diese Box verändert.

```
\par Es folgt eine V-Box
     \vbox{\hsize=0.50\hsize
           \noindent
           Dies ist eine typische Textzeile.
           Dies ist eine typische Textzeile.
           Dies ist eine typische Textzeile.}
      und noch was.\par
```

liefert

Dies ist eine typische Textzeile. Dies ist eine
typische Textzeile. Dies ist eine typische Text-
Es folgt eine V-Box zeile. und noch was.

Bisher wurde die \vbox nach untersten bzw. bei \vtop nach der obersten Zeile beim Satz ausgerichtet. Es ist auch möglich, eine \vbox bezüglich ihrer Höhe zu zentrieren. Der Befehl dazu ist \vcenter. Dieser ist allerdings nur im *mathematical mode* erlaubt. Das stört aber nicht weiter, wie man an folgendem Beispiel sieht:

```
\par Es folgt eine V-Box
     $\vcenter{\vbox{\hbox{erste H-Box}\hbox{‘zweyte’ H-Box}}}$
     und noch was.
```

liefert

Es folgt eine V-Box erste H-Box
‘zweyte’ H-Box und noch was.

8.4 Box-Register

Wichtig ist für die praktische Verwendung die Möglichkeit, Boxen zunächst zu erstellen, aufzuheben und erst später wieder auszugeben. Es gibt nun im TEX-Programm 256 Box-Register mit den Nummern 0 bis 255. Davon werden einige allerdings schon vom 'plain-TEX' selbst benutzt. Laut Konvention sind aber die Register von 0 bis 9 frei, werden aber unter Umständen von einigen TEX-Makros auch mitbenutzt. Im weiteren kann man sich aber durch zum Beispiel `\newbox\meinebox` die Nummer der nächsten garantiert freien Box auf "`\meinebox`" zuteilen lassen. Der Befehl `\setbox` erlaubt die Zuweisung einer folgenden `\hbox` oder `\vbox` auf ein solches Box-Register:

```
\setbox0=\hbox{Testdaten}
\setbox1=\vbox{\hbox{eins}\hbox{zwei}\hbox{drei}}
\newbox\meinebox
\setbox\meinebox=\hbox{Text für meine Box!}
```

Damit wird in den Box-Registern Information — jeweils als `\hbox` oder `\vbox` — abgelegt.

Auch hier gilt übrigens der Grundsatz mit den Klammerstrukturen. Boxen, die innerhalb von Klammergebirgen mit `\setbox` verändert werden, haben nach dem Verlassen einer inneren Struktur wieder den alten Wert, der durch den äußeren `\setbox` Befehl belegt wurde!

Durch ein `\global` vor der `\setbox`-Anweisung wird jedoch der Wert in allen äußeren Ebenen auch geändert. Die Anweisungen

```
\setbox0=\hbox{Vorher}
{\global\setbox0=\hbox{Alles getan!}}
```

hinterlassen in der Box 0 den Inhalt "`\hbox{Alles getan!}`".

Hier sei noch einmal darauf verwiesen, daß das Ergebnis einer `\setbox`-Anweisung eine Box ist, für die es beim weiteren Satz nicht mehr relevant ist, ob sie durch Zuweisung mittels `\hbox` oder `\vbox` entstanden ist. `\hbox` und `\vbox` haben nur Wirkung auf das *Innere* dieser Box!

8.5 Box-Ausgaben

Nachdem nun mittels eines `\setbox`-Befehls der Inhalt einer Box gespeichert werden kann, sind natürlich die Ausgabebefehle von Interesse; denn neben Speicherung steht der Abruf. Dabei sind folgende Eigenschaften bei der Weiterverarbeitung einer Box zu beachten:

- Ausgabe als *Kopiervorgang* unter Erhaltung der Information in der betreffenden Box
- Ausgabe als *Transportvorgang*: Anschließend ist die Quellbox leer.
- Ausgabe mit "*Entblättern*": Das heißt, die äußere Box-Klammer entfällt und die Box wird wieder in die Unterboxen, aus denen sie besteht, zerlegt.
  Beim Entblättern muß allerdings angegeben werden, welche mögliche äußere Struktur ("`\vbox`" oder "`\hbox`") entfernt werden soll.

Aus den Kombinationen dieser verschiedenen Möglichkeiten ergeben sich dann eine Reihe von Befehlen.

Ausgabe

`\box` Der Befehl "`\box`*n*" gibt den Inhalt des angegebenen Box-Registers aus. Die Quellbox ist anschließend leer.

```
\setbox0=\hbox{Anton Meier}   \box0
```

hat die Wirkung wie ein "`\hbox{Anton Meier}`". Bei der Verwendung von `\box` braucht keine Rücksicht auf die Struktur des Box-Registers, das heißt `\vbox` oder `\hbox`, genommen werden. Wird der `\box` Befehl in einer inneren Gruppe angewendet, so ist die Quellbox auch nach dem Verlaufen einer Gruppe leer, wenn die letzte Belegung mit `\setbox` in der äußeren Gruppe erfolgte.

```
\setbox0=\hbox{Anton} {\box0}
\setbox1=\hbox{Willi} {\setbox1=\hbox{Karl}\box1}
```

hinterlassen `\box0` leer und `\box1` mit dem Inhalt `\hbox{Willi}`.

Kopieren

`\copy` Der Befehl "`\copy`*n*" gibt den Inhalt des angegebenen Box-Registers aus. Die Quellbox enthält anschließend noch immer den gleichen Inhalt.

```
\setbox0=\hbox{Anton Meier}   \copy0
```

hat die Wirkung wie ein "`\hbox{Anton Meier}`". Es braucht auch hier keine Rücksicht auf den Inhalt des Box-Registers genommen werden.

Ausgeben und Kopieren mit Entblättern

`\unhbox` Durch diesen Befehl wird der Inhalt der angegebenen Box ausgegeben, jedoch die äußere `\hbox` entfernt. Der Inhalt der Quellbox ist anschließend eine leere Box.

```
\setbox0=\hbox{Anton Meier}
\setbox1=\hbox{\hbox{Anton }\hbox{Meier}}
\unhbox0  % entspricht  'Anton Meier'
\unhbox1  % entspricht  '\hbox{Anton }\hbox{Meier}'
```

Diese Befehle haben nach "`\unhbox0`" die Wirkung wie die einfache Eingabe "`Anton Meier`", beziehungsweise nach "`\unhbox1`" wie "`\hbox{Anton }\hbox{Meier}`". Es wird also eine Verklammerungsebene entfernt.

`\unvbox` Durch diesen Befehl wird der Inhalt der angegebenen Box ausgegeben, jedoch die äußere `\vbox` entfernt. Der Inhalt der Quellbox ist anschließend leer.

`\unhcopy` Die Wirkung ist bei der Ausgabe wie bei "`\unhbox`". Es bleibt jedoch der Inhalt der Quellbox erhalten.

`\unvcopy` Die Wirkung ist bei der Ausgabe wie bei "`\unvbox`". Es bleibt jedoch der Inhalt erhalten.

8.6 Box-Dimensionen

Eine Box hat immer eine bestimmte Größe. TEX kennt zu jeder Box 3 Größen. Diese sind ihre Breite *width* — `\wd`, ihre Höhe *height* — `\ht` und ihre Tiefe *depth* — `\dp`. Die Höhe und Tiefe werden immer von der Grundlinie aus gerechnet. Zeichen ohne Unterlängen besitzen die Tiefe Null.

Die natürliche Größe *(natural width)* einer Box wird durch den Umfang der Information bestimmt, die in ihrem Inneren steht. Daneben gibt es noch die Möglichkeit, die Breite einer `\hbox` und die Höhe einer `\vbox` explizit mitanzugeben. Durch den Befehl `\hbox to 2cm{...}` oder `\vbox to 2cm{...}` wird festgelegt, wie groß die Boxen sein sollen. Enthält eine solche Box kein dynamisches Material, dies sind `\hfil`, `\hfill` und `\hss` für eine `\hbox` und `\vfil`, `\vfill` und `\vss` für eine `\vbox`, so ist die Box meist immer zu groß oder zu klein. TEX wird dann eine Fehlermeldung ausgeben.

Neben der externen Vorgabe einer festen Größe für eine Box kann auch angegeben werden, um welchen Betrag sie größer als ihre natürliche Größe sein soll. Durch

```
\hbox spread 1cm {...}
\vbox spread 2cm {...}
```

wird diese Box um 1 cm größer ausfallen, als es durch den normalen inneren Text bedingt wäre. Es muß dann allerdings auch dynamischer Leerraum im Innern der Box vorhanden sein.

Die Größen der Boxen, die mittels `\setbox` gespeichert sind, lassen sich durch Angabe der Box-Nummer hinter `\wd`, `\dp` oder `\ht` abfragen und auch verändern.

Breite	*Höhe*	*Tiefe (Unterlänge)*
`\wd0`	`\ht0`	`\dp0`
`\wd1`	`\ht1`	`\dp1`
nach "`\newbox\meinebox`":		
`\wd\meinebox`	`\ht\meinebox`	`\dp\meinebox`

Damit besteht die Möglichkeit, auch einmal die Breite eines Textes auszumessen:

```
\setbox0=\hbox{\it Rainer Maria Rilke}
```

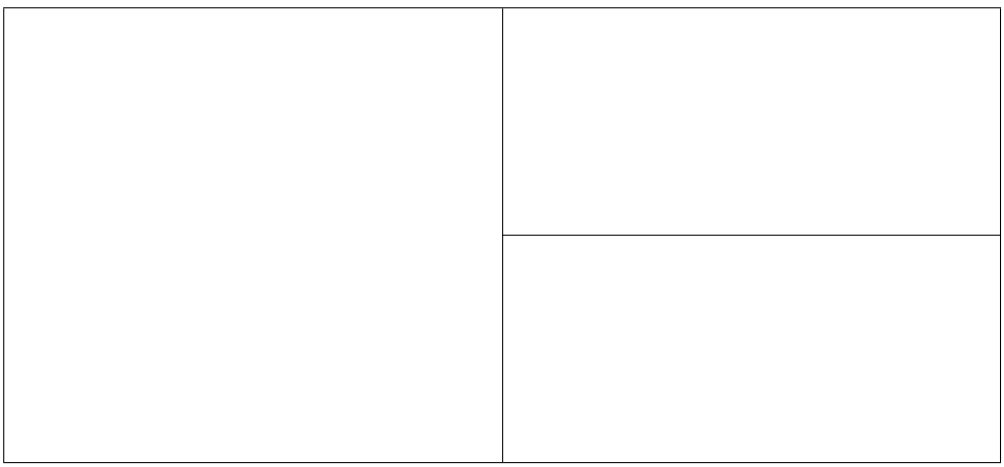
Anschließend enthält `\wd0` die genaue Breite des Textes, der durch die Eingabe `\it Rainer Maria Rilke` entstanden ist. Diese Breite kann weiterverwendet werden, zum Beispiel in `\parindent=\wd0` oder `\hskip\wd0`.

Mit Hilfe fester Box-Größen kann man dann auch so etwas wie Formulare setzen, bei denen die Bestandteile zusammengesetzt werden.

Zur Verdeutlichung einer solchen Technik soll im folgenden Beispiel einmal ein solches Formular erzeugt werden. Die Ausgabe soll 30 % einer normalen Seite hoch sein (0.3`\vsize`), wobei diese halbe Seite noch mal in zwei Hauptspalten unterteilt ist. Jede dieser Spalten ist dann 0.5`\hsize` breit. Die rechte Spalte soll allerdings noch mal in zwei untereinanderliegende Boxen zerlegt werden. In den großen linken Kasten kommt nach allen Seiten zentriert ein Titel. Die beiden rechten Kästchen erhalten normalen Fließtext, wobei der obere etwas schmaler gesetzt werden soll.

```
\def\MeinFormular#1#2#3{\par\hrule\noindent
     \vbox to 0.3\vsize                      % linker Kasten
           {\hsize=0.5\hsize                 % :
            \vfill                           % :
            \hbox to \hsize                  % :
              {\hfill#1\hfill}               % :
            \vfill}                          % :
     \vbox to 0.3\vsize                      % rechter Kasten
           {\hsize=0.5\hsize                 % :
            \hbox                            % :  rechts oben
              {\vbox to 0.15\vsize           % :  :
                 {\narrower                  % :  :
                  \par\noindent#2\vfill}}    % :  :
            \hbox                            % :  rechts unten
              {\vbox to 0.15\vsize           % :  :
                 {\par\noindent#3\vfill}}    % :  :
           }                                 % :  :
     \hrule}
```

Dieses Formular hat dann folgende Gestalt:



Der Aufruf

```
\MeinFormular{\bf T I T E L}%
  {\it Dies ist eine obere Zeile. Dies ist eine obere Zeile.
       Dies ist eine obere Zeile. Dies ist eine obere Zeile.}
  {\bf Dies ist eine untere Zeile. Dies ist eine untere Zeile.
       Dies ist eine untere Zeile. Dies ist eine untere Zeile.}
```

liefert

Dies ist eine obere Zeile. Dies ist eine obere Zeile. Dies ist eine obere Zeile. Dies ist eine obere Zeile.

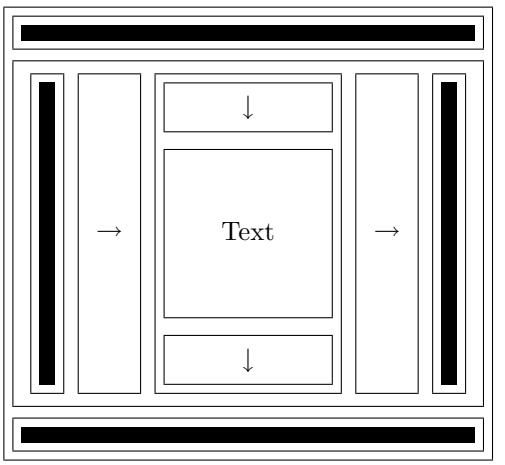
T I T E L

Dies ist eine untere Zeile. Dies ist eine untere Zeile. Dies ist eine untere Zeile. Dies ist eine untere Zeile.

8.7 Einrahmungen

Nach der nun vorhandenen Kenntnis der `\hbox`- und `\vbox`-Befehle können jetzt auch Makros für "Einrahmungen" von Textstücken erzeugt werden.
Im weiteren soll mit dem Grundmakro `\rahmen` gearbeitet werden:

```
\def\rahmen#1#2{
   \vbox{\hrule
         \hbox
           {\vrule
            \hskip#1
            \vbox{\vskip#1\relax
                  #2%
                  \vskip#1}%
            \hskip#1
            \vrule}
         \hrule}}
%
% #1 ist der Abstand der
%    Umrahmungslinie
% #2 ist die zu umrahmende
%    Information
```



Box-Struktur von `\rahmen`

Das Makro `\rahmen` erzeugt eine Boxstruktur entsprechend der oberen Abbildung. `\hrule` und `\vrule` sind durch dicke Stricke markiert, `\hskip` und `\vskip` durch "→" und "↓". Die erzeugten `\vbox`- und `\hbox`-Strukturen sind jeweils selbst noch einmal durch Kästchen verdeutlicht.

Der Befehl `\rahmen` bildet in der äußeren Ebene eine große `\vbox`. Diese `\vbox` besteht aus 3 Hauptbestandteilen. Diese sind eine obere `\hrule`, eine mittlere `\hbox` und eine untere `\hrule`. Die mittlere `\hbox` besteht aus fünf Unterboxen: der ersten `\vrule`, etwas Leerraum (`\hskip`), einer weiteren `\vbox`, etwas Leerraum (`\hskip`) und der rechten `\vrule`. Die innerste `\vbox` besteht aus oberem Leerraum (`\vskip`), der eigentlichen Information und unterem Leerraum (`\vskip`).

Nach dieser langen Erläuterung nun einige Beispiele:

```
$$ \rahmen{0.5cm}{\hsize=0.7\hsize % kurze Zeilen
     \noindent\bf To read means to obtain meaning from words
                 and legibility is that quality which enables
                 words to be read easily, quickly, and accurately.
     \smallskip
     \hfill \it John Charles Tarr} $$ % Dollars zur Zentrierung
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

Mehrfache Rahmen sind so auch möglich:

```
$$\rahmen{0.1cm}%
   {\rahmen{0.5cm}{\hsize=0.7\hsize % kurze Zeilen
     \noindent\bf To read means to obtain meaning from words
                 and legibility is that quality which enables
                 words to be read easily, quickly, and accurately.
     \smallskip
     \hfill \it John Charles Tarr}}$$
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

Die Information, die eingerahmt wird, steht in einer `\vbox`. Durch die Angaben `\hsize=0.7\hsize` wird die Länge einer Zeile auf 70 % ihres alten Wertes reduziert. Durch den Befehl `\noindent` wird aus dem *vertical mode* in den *horizontal mode* gewechselt. Damit wird der Absatzumbruch gestartet. Der Absatz wird durch den Befehl `\smallskip` beendet. Durch `\hfill` wird der Absatzumbruch — *horizontal mode* — neu gestartet, wobei hier nur eine Zeile mit rechtsbündigem Text erzeugt wird.

Die rechtsbündige Ausgabe kann auch durch ein "`\rightline{\it John Charles Tarr}`" erreicht werden.

Das \rahmen-Makro kann leicht um einen Parameter erweitert werden, der die Dicke der Striche angibt:

```
\def\Rahmen#1#2#3{                %
     \vbox{\hrule height#2        %  #1 ist der Abstand
          \hbox{\vrule width#2    %
                \hskip#1          %  #2 ist die Strichdicke
                \vbox{\vskip#1{}  %
                      #3          %  #3 ist die Information
                      \vskip#1}   %
                \hskip#1
                \vrule width#2}
          \hrule height#2}}
```

Die Anwendung einer doppelten Umrahmung

```
$$\Rahmen{0.1cm}{1.5pt}%
   {\Rahmen{0.5cm}{0.4pt}{\hsize=0.7\hsize % kurze Zeilen
     \noindent\bf To read means to obtain meaning from words
                 and legibility is that quality which enables
                 words to be read easily, quickly, and accurately.
     \smallskip
     \hfill \it John Charles Tarr}}$$
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

Beide Makros \rahmen und \Rahmen behandeln die zu umrahmende Information mit einer \vbox. Dies bedeutet, wenn nur ein einzelnes Wort eingerahmt werden soll, muß dieses mit einer zusätzlichen \hbox versehen werden. Hier wurde ein Befehl \Kiste{Wort} aufgerufen, der durch \def\Kiste#1{\rahmen{1.5pt}{\hbox{#1}}} definiert ist.

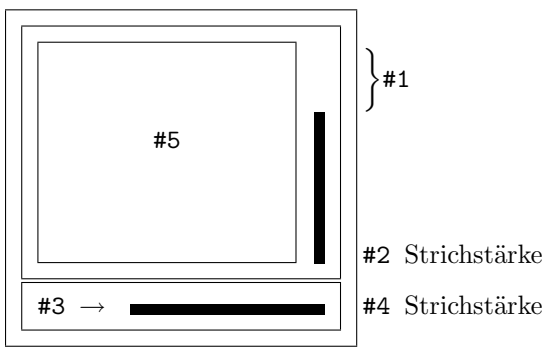
Anmerkung: Einrahmungen können ihre sinnvolle Funktion besitzen, ich möchte jedoch davor warnen, dieses optische Gestaltungsmittel zu überstrapazieren. Insbesondere doppelte Einrahmungen bilden zwar einerseits einen Augenfänger, solche Stilmittel sollen andererseits jedoch nur selten verwendet werden. Hierbei soll allerdings zwischen Tabellenlinien und Rähmchen für Texte unterschieden werden. In Tabellen sind strukturierende Linien häufig eine Notwendigkeit.

8.8 Blöcke und Schatten

Die vorstehenden Makros können leicht verändert werden, um einen Schatten- oder Blockeffekt zu erzielen. Das Basismakro für beide Effekte zieht eine horizontale Linie unterhalb und eine vertikale neben das Objekt. Werden die beiden Linien am Anfang etwas kürzer und am Ende etwas länger gesetzt, erhält man den gewünschten optischen Effekt. Die Strichstärke ist ein zusätzlicher Parameter dieses Makros:

```
\def\BaseBlock#1#2#3#4#5{%          %  #1 Versatz der Senkrechten
   \vbox{\setbox0=\hbox{#5}%        %  #2 Strichstärke Senkrechte
         \offinterlineskip          %  #3 Versatz der Horizontalen
         \hbox{\copy0               %  #4 Strichstärke Horizontale
              \dimen0=\ht0          %  #5 Information
              \advance\dimen0 by -#1
              \vrule height \dimen0 width#2}%
         \hbox{\hskip#3\dimen0=\wd0
              \advance\dimen0 by -#3
              \advance\dimen0 by #2
              \vrule height #4 width \dimen0}%
         }}%
```

Die Information in Parameter #5 wird in eine Hilfsbox gesetzt, um mit ihrer Größe die Längen der Striche zu erhalten. Als Ergebnis erhält man eine "`\vbox`" mit der folgenden Struktur:



Das Makro `\BaseBlock` kann nun sehr leicht verwendet werden, um den Schatteneffekt zu erzeugen:

```
\def\Schatten#1{\BaseBlock{4pt}{2pt}{4pt}{6pt}{#1}}
```

Die verwendeten Parameter erzeugen 4 pt und 6 pt dicke Linien. Der vertikale und horizontale Offset sind gleich groß. Da das `\BaseBlock` Makro keinen umgebenden Rahmen erzeugt, wird es einfach mit dem schon bekannten `\rahmen` Makro kombiniert:

```
$$\Schatten{\rahmen{0.5cm}{\hsize=0.7\hsize
     \noindent\bf To read means to obtain meaning from words
               ...
          \hfill \it John Charles Tarr}}$$
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

Im Gegensatz zum Beispiel im Abschnitt 6.6 werden hier die Schatten in einem einzigen Stück erzeugt. Im Abschnitt 6.6 wurde `\halign` verwendet, wobei durch viele `\vrule` Stückchen die vollständige Linie gebildet wurde.

Um einen Blockeffekt zu erreichen, werden einfach sehr viele dünne Linien mit wachsendem Versatz kombiniert. Dies wird mit Hilfe des `\loop` Befehls konstruiert. In dem folgenden Beispiel besitzen die Linien eine Strichstärke von 0.4 pt und einen gleich großen Offset. Der Parameter "`#2`" im Makro `\LoopBlock` bestimmt, wie viele Linien nebeneinander gesetzt werden sollen. Die Befehle `\begingroup` und `\endgroup` bilden innerhalb des Makros eine Gruppe, so daß alle Registerveränderungen lokal bleiben. Sie könnten für die Aufgabe auch durch "`{`" und "`}`" ersetzt werden.

```
\def\LoopBlock#1#2{%
\begingroup
  \dimen2=0.4pt    % Inkrement / Linienabstand
  \def\doblock{%
    \setbox2\BaseBlock
        {\count1\dimen2}{0.4pt}{\count1\dimen2}{0.4pt}{\box2}}%
  \setbox2=\vbox{#1}%  Anfangsinformation
  \count1=0
  \loop
      \advance\count1 by 1
      \doblock
      \ifnum\count1<#2
  \repeat
  \box2
\endgroup}
```

Der Einfachheit halber wird dieses Makro in einem weiteren Makro versteckt:

```
\def\Block#1{\LoopBlock{#1}{10}}
```

Die Eingabe

```
$$\Block{\rahmen{0.5cm}{\hsize=0.7\hsize
     \noindent\bf To read means to obtain meaning from words
                 and legibility is that quality which enables
                 words to be read easily, quickly, and accurately.
              \smallskip
             \hfill \it John Charles Tarr}}$$
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

Wenn der später zu verwendende Drucker eine genügende Auflösung besitzt, so ist mit einer kleiner Änderung sogar ein Grauton möglich:

```
\def\LoopGrauBlock#1#2{%
\begingroup
  \dimen2=0.4pt    % Inkrement / Linienabstand
  \def\leer{\setbox2=\vbox                                % <<< neu
              {\hbox{\box2\hskip\dimen2}\vskip\dimen2}}% <<< neu
  \def\doblock{%
    \setbox2\BaseBlock
        {\count1\dimen2}{0.4pt}{\count1\dimen2}{0.4pt}{\box2}}%
  \setbox2=\vbox{#1}%  Anfangsinformation
  \count1=0
  \loop
      \advance\count1 by 2 % <<< geändert
      \leer                % <<< neu
      \doblock
      \ifnum\count1<#2
  \repeat
  \box2
\endgroup}
%
\def\GrauBlock#1{\LoopGrauBlock{#1}{10}}
```

Die Eingabe

```
$$\GrauBlock{\rahmen{0.5cm}{\hsize=0.7\hsize
     \noindent\bf To read means to obtain meaning from words
                 and legibility is that quality which enables
                 words to be read easily, quickly, and accurately.
              \smallskip
             \hfill \it John Charles Tarr}}$$
```

liefert

To read means to obtain meaning from words and legibility is that quality which enables words to be read easily, quickly, and accurately.

John Charles Tarr

8.9 Dynamischer Leerraum

Die bisher erwähnten Skip-Befehle für horizontalen und vertikalen Platz

horizontaler Leerraum	*vertikaler Leerraum*
`\quad`	`\smallskip`
`\qquad`	`\medskip`
	`\bigskip`
`\hskip` *Länge*	`\vskip` *Länge*

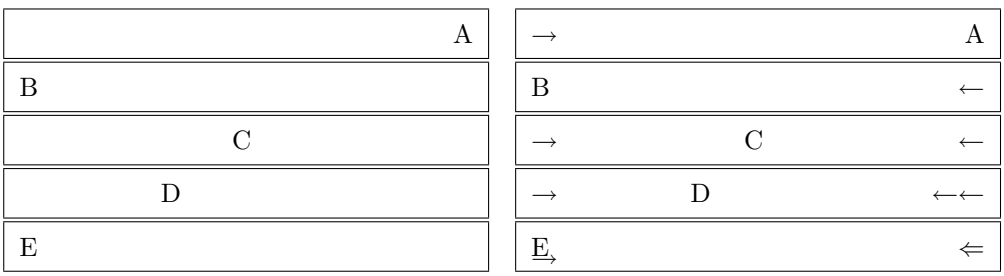
ließen immer Leerraum von einem *festen* Betrag. Daneben gibt es noch Befehle, die *dynamischen* Leerraum erzeugen. Dieser kann — je nach Befehl — sich dehnen oder auch schrumpfen. Verwendet wird solcher dynamischer Leerraum für Ausrichtungen, wie Zentrierung oder um Information an den linken, rechten, oberen oder unteren Rand einer Box zu setzen.

horizontal	*vertikal*	*Bedeutung*
`\hfil`	`\vfil`	dehnbarer Leerraum
`\hfill`	`\vfill`	dehnbarer Leerraum, aber stärker als `\hfil`, `\vfil`
`\hss`	`\vss`	dehnbar und schrumpfbar

Beispiele

```
\rahmen{0.2cm}{\hbox to 6cm{\hfil A}}            % nach rechts
\rahmen{0.2cm}{\hbox to 6cm{B \hfil}}            % nach links
\rahmen{0.2cm}{\hbox to 6cm{\hfil C \hfil}}      % in der Mitte
\rahmen{0.2cm}{\hbox to 6cm{\hfil D \hfil\hfil}} % 1 zu 2 geteilt
\rahmen{0.2cm}{\hbox to 6.4cm{\hfil E \hfill}}   % nach links !
```

liefert mit Pfeilmarkierung:



Daneben sind noch "`\hfilneg`" und "`\vfilneg`" vorhanden, die ein vorangehendes "`\hfil`" bzw. "`\vfil`" wieder aufheben.

Überlappungen

Durch die Befehle `\hss` und `\vss` sind auch solche Dinge wie Überdrucken möglich. Die beiden folgenden TEX-Standardmakros `\rlap` und `\llap` machen genau dies:

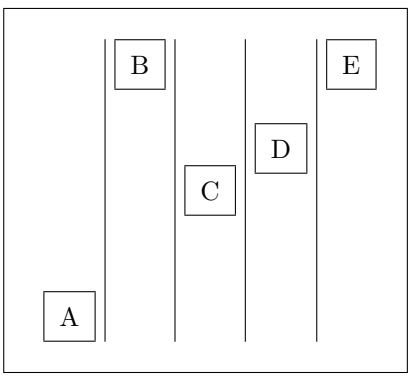
```
\def\llap#1{\hbox to 0pt{\hss#1}}
\def\rlap#1{\hbox to 0pt{#1\hss}}
```

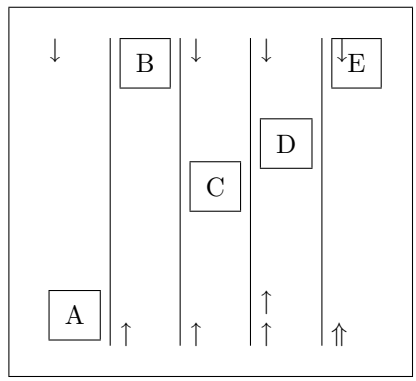
`\rlap` druckt den Text nach rechts und geht dann wieder zurück und `\llap` druckt den Text nach links hinausragend.

`1\llap{/}2` erzeugt 12, bzw. `1\rlap{/}2` erzeugt 12.

In der gleichen Weise kann vertikaler dynamischer Leerraum in einer `\vbox` verwendet werden. Im folgenden Beispiel wird die Wirkungsweise von `\vfil` und `\vfill` demonstriert:

```
\def\testbox#1{\rahmen{0.2cm}{\hbox{#1}}}
\rahmen{0.4cm}{\hbox{
          \vbox to 4cm{\vfil\testbox A}
 \vrule\  \vbox to 4cm{\testbox B\vfil}
 \vrule\  \vbox to 4cm{\vfil \testbox C \vfil}
 \vrule\  \vbox to 4cm{\vfil \testbox D \vfil\vfil}
 \vrule\  \vbox to 4cm{\vfil \testbox E \vfill}}}
```





8.10 Box-Bewegungen

Bisher wurde die Information im Innern ihrer umgebenden Box bewegt. Es ist jedoch auch möglich, eine fertige Box bei der späteren Ausgabe nach oben, unten bzw. links und rechts zu verschieben. Dabei ist diese Verschiebung immer senkrecht zur aktuellen Satzrichtung. Dies bedeutet, im horizontalen Arbeitsmodus (`\hbox` oder Absatzumbruch) kann eine Box mittels der Befehle "`\raise`" sowie "`\lower`" nach oben und unten bewegt werden.

```
\par Das folgende Wort steht \raise3pt\hbox{oben}
     oder auch \lower3pt\hbox{unten}!
```

liefert

Das folgende Wort steht $^{\text{oben}}$ oder auch $_{\text{unten}}$!

Die Befehle besitzen die Syntax

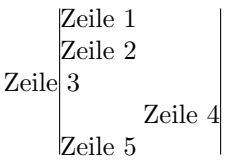
`\raise` *dimension box-angabe*
`\lower` *dimension box-angabe*

Eine Angabe von "`\raise3pt\hbox{...}`" wirkt wie "`\lower-3pt\hbox{..}`".

Im vertikalen Arbeitsmodus bewirken die Befehle “\moveleft” und “\moveright” mit gleicher Eingabesyntax wie “\raise” Verschiebungen nach links und rechts. Die Befehle lassen den Bezugspunkt der umgebenden Box unverändert, allerdings kann bei Bewegungen nach rechts die umgebende Box breiter werden.

```
$$\hbox {\vrule \vbox {\hbox{Zeile 1}
                       \hbox{Zeile 2}
                       \moveleft  7.5mm \hbox{Zeile 3}
                       \moveright  11mm \hbox{Zeile 4}
                       \hbox{Zeile 5}}%
         \vrule}$$
```

erzeugt



In der Praxis findet man jedoch häufig Konstruktionen, in denen auch einmal Information *neben* die Box gesetzt wird. Es sind beim Satz und der Weiterverarbeitung einfach zwei Dinge zu unterscheiden: die *logische* Größe einer Box (Höhe, Tiefe und Breite), welche die Verarbeitung dieser Box in ihrer Umgebung regelt, und die tatsächlich gesetzte Information. Diese braucht nämlich *nicht* im Innern ihrer zugehörigen Box zu liegen. Beispiele für solche Anwendungen sind die oben schon erwähnten Befehle “\llap” und “\rlap” für Überlappungen.

Ein Beispiel für eine etwas komplexere Konstruktion ist ein Makro, das genau neben die Textzeile, aus der es aufgerufen wird, eine Randmarkierung setzt. Hier wird \Randmarkierung gesetzt, um neben die aktuelle Zeile einen Pfeil zu setzen. Auch ⇐
dabei sind Positionierungen notwendig, die außerhalb der Boxgrenzen liegen.

Dazu muß der Befehl “\vadjust{...}” kurz erläutert werden. Die als Parameter angegebene vertikale Satzinformation wird *nach* der aktuellen Zeile, in der der \vadjust-Befehl steht, eingefügt. So ist hier ein \vadjust{\smallskip} eingegeben worden, so daß die folgende Zeile mit etwas größerem Zeilenabstand gesetzt wird. Der Vorteil bei diesem Verfahren liegt in dem Sachverhalt, daß der Umbruch um die Einfügung sozusagen herumläuft. Durch “\vadjust{\vskip 5cm}” kann auf diese Weise Platz für eine Illustration gelassen werden.

Das Markierungsmakro hat dann folgende Gestalt:

```
\def\Randmarkierung{\vadjust{\vbox to 0pt
     {\vss
      \hbox to \hsize{\hskip\hsize
                      \quad
                      $\Leftarrow$\hss}
      \vskip3.5pt}}}
```

Wie der Leser sicher festgestellt hat, sind Bewegungen mit *leeren* Boxen, das heißt mit Boxen, bei denen alle Längen Null sind, hinterher am einfachsten zu handhaben. Eine

solche Box läßt sich überall ohne Verschiebungen in der Zielumgebung unterbringen und die Positionierungen geschehen stets bezüglich des so definierten Ursprungspunktes.

Wenn so mit untereinander gesetzten Boxen hantiert wird, ist zu beachten, daß auch hier die Kontrolle der Abstände aufeinanderfolgender Boxen (Zeilen) bezüglich nachstehender Werte erfolgt.

`\baselineskip`	bildet den normalen Abstand der Grundlinien im Fließtext, der mit 12 pt vorbelegt ist.
`\lineskiplimit`	definiert die Abstandsschranke zweier untereinanderstehender Boxen. Wird dieser unterschritten, so wird `\lineskip` eingesetzt.
`\lineskip`	setzt den Mindestabstand zweier untereinanderstehender Boxen. `\lineskip` ist vorbelegt mit 1 pt.

Sollen entstandene Boxen direkt untereinander ohne zusätzlichen Leerraum zwischen ihnen gesetzt werden, so sind diese Werte entsprechend zu korrigieren.

Der Befehl "`\offinterlineskip`" schaltet die Kontrolle für den Abstand ab. Die Boxen können vertikal dicht an dicht gepackt werden.

Durch "`\nointerlineskip`" wird dagegen dies nur für eine einmalige Ausführung gesetzt, anschließend gelten wieder die vorher gesetzten Mindestabstände.

Künstliche Grundlinien

Wie vorher schon dargestellt, werden Boxen, wenn sie nebeneinander gesetzt werden, bezüglich ihrer Grundlinien ausgerichtet. Bei "`\vbox`" wird diese aufgrund der untersten Box und bei "`\vtop`" aufgrund der obersten inneren Box bestimmt. Oft ist in Makros unbekannt, wo diese Grundlinie liegen wird. Es kann ja sein, daß eine große Schrift angewählt wird und dadurch eine Grundlinie relativ zum Boxanfang tiefer zu liegen kommt.

Durch einen Befehl "`\hrule height0pt`" kann nun an den Anfang oder das Ende einer Box eine solche virtuelle Grundlinie gesetzt werden. Damit wird die Ausrichtung bezüglich dieses nicht sichtbaren Striches normiert.

Auf der anderen Seite ist es häufig notwendig, durch den Befehl `\strut` die erste oder letzte Zeile einer `\vbox` mit der Höhe und Unterlänge einer Standardzeile zu besetzen. Durch ein `\strut` wird die minimale Höhe einer Zeile auf 8.5 pt und die minimale Unterlänge auf 3.5 pt gesetzt.

8.11 Box-Teilausgaben

Eine Möglichkeit, nur einen Teil einer `\vbox` auszugeben, bietet der Befehl `\vsplit`. Er besitzt die Syntax `\vsplit` *Boxnummer* `to` *Länge*. Mit ihm kann aus einer `\vbox` der obere Teil mit einer gewünschten Länge abgespalten und ausgegeben werden. Dabei wird wie beim Seitenumbruch nach einer möglichst guten Umbruchstelle an der gewünschten Position gesucht. Der nicht ausgegebene Rest bleibt in der Quellbox erhalten.

Damit lassen sich zum Beispiel leicht zweispaltige Ausgaben, die auf einer Seite bleiben, realisieren. Zunächst wird der gesamte Text in einer Hilfsbox eingesammelt und anschließend geteilt.

Beispiel:

```
\setbox0=\vbox{\hsize=0.4\hsize
               \it
               \obeylines
               2 Zwiebeln
               5 E\ss l\"offel \"Ol
               40 g Mehl
               $\it 1/4$ l Bratensaft (W\"urfel)
               $\it 1/8$ l saure Sahne
               Salz
               Pfeffer
               1 E\ss l\"offel Zitronensaft
               2 Gew\"urzgurken
               100 g Champignons (Dose)
               500 g Rinderfilet}
```

Damit enthält die Box 0 nun Zeile für Zeile den Text. Insbesondere enthält das Box-Register `\ht0` die Höhe der so entstandenen Box. Mit ein wenig Arithmetik

```
\dimen1=0.5\ht0
\advance\dimen1 by 0.5\baselineskip
```

ergibt sich die gewünschte Umbruchlänge. Die Addition des halben Zeilenabstandes geschieht, um bei einer ungeraden Zeilenzahl links eine Zeile mehr zu besitzen. Nach

```
\splittopskip=0pt
\setbox1=\vsplit 0 to \dimen1
```

ist die obere Hälfte nach Box-Register 1 ausgegeben. Damit lassen sich die beiden Teile leicht zusammenstellen. Der Parameter "`\splittopskip`" wird vor der `\vsplit`-Operation auf 0 pt gesetzt. In gleicher Weise wie beim normalen Seitenumbruch wird an der Umbruchstelle ein vertikaler Skip gesetzt, der den Abstand der ersten Grundlinie zum Seitenanfang bestimmt. Beim Seitenumbruch ist dies "`\topskip`", hier ist es der Parameter "`\splittopskip`".

Die "`\hrule`" der Höhe 0 pt wird gesetzt, um eine leere Anfangsbox zu erhalten, an der das "`\vtop`" ausgerichtet wird.

```
\hbox to \hsize{\hfil\vtop{\hrule height 0pt
                           \box1}%
                \hfil\vtop{\hrule height 0pt
                           \box0}%
                \hfil}
```

liefert in zweispaltiger Form die Zutaten zu "Bœuf Stroganoff":

2 Zwiebeln
5 Eßlöffel Öl
40 g Mehl
1/4 l Bratensaft (Würfel)
1/8 l saure Sahne
Salz
Pfeffer
1 Eßlöffel Zitronensaft
2 Gewürzgurken
100 g Champignons (Dose)
500 g Rinderfilet

Noch ein Hinweis: Die `\vsplit`-Operation arbeitet wie der normale Seitenumbruch. Damit erhält man auch die typischen Fehlermeldungen, wie etwa "`underfull vbox..`", wenn man etwa mehr abzuspalten versucht, als in der Quellbox enthalten ist.

8.12 Zähl- und Längenregister

Neben den oben besprochenen Box-Registern kennt das TEX-Programm noch einige andere Register, die zum Teil auch schon verwendet wurden. Für alle Register gilt: Es gibt jeweils 256 Stück, die von 0 bis 255 durchnumeriert werden. Die Register 0 bis 9 gelten als frei für eine lokale Benutzung als temporäre Hilfsregister. Sie werden aber auch schon mal von den plain-TEX-Makros lokal benutzt. Daher sollte man sich für spezielle Anwendungen mit Hilfe der zugeordneten `\new...`-Befehle ein garantiert freies Register reservieren.

`\count0...` numerisches Register für ganze Zahlen:
Als Konvention enthält `\count0` die Seitennumerierung. Diese kann auch unter dem Namen `\pageno` referiert werden.

`\dimen0 ...` Längenregister, in dem Dimensionsangaben gespeichert werden. Diese Register verhalten sich wie die schon bekannten Register `\ht`, `\dp` und `\wd`, die ja einzelnen Box-Registern zugeordnet sind.
Hier können einfache Längenangaben unabhängig von den verschiedenen Einheiten cm, mm, pt usw. gespeichert werden.

`\skip0 ...` skip-Register, die ähnlich wie ein `\dimen`-Register arbeiten; jedoch dürfen diese dynamische skip-Anteile wie "1 cm minus 0.5 cm plus 0.5 cm" enthalten.

`\muskip0 ...` sind skip-Register für den Gebrauch im Mathematiksatz. Hier können nur Längenangaben mit der Einheit "mu" verwendet werden.

Reservierung	*Zuweisung / Änderung*	`Ausgabe / Abfrage`
`\newcount\mycount`	`\count1=17`	`\number\count1`
	`\mycount=4`	`\number\mycount`
	`\mycount=\count2`	`\ifnum\mycount>1 ...`
	`\advance\mycount by 1`	
`\newdimen\mydimen`	`\dimen1=17cm`	`\hbox to \dimen1 ...`
	`\mydimen=4cm`	`\vbox to \mydimen ...`
	`\advance\mydimen by 1pt`	`\ifdim\dimen1 > 1cm`
	`\mydimen=0.5\mydimen`	`\ifdim\mydimen < 1 pt`
`\newskip\myskip`	`\skip1=5cm`	`\vskip\skip1`
	`\skip1=5cm minus 1cm`	`\hskip\skip1`
	`\myskip=1pt plus 1pt`	`\vskip\myskip`
	`\skip1=2\myskip`	
	`\advance\myskip by 1pt`	
`\newmuskip\mymuskip`	`\muskip1=5mu`	`\mskip\muskip1`
	`\mymuskip=5mu minus 1mu`	`\mskip\mymuskip`

8.13 Token Register

Im weiteren existieren 256 *token register*, die ähnlich einer Makrodefinition verwendet werden können. Sie werden spezifiziert durch `\toks0` bis `\toks255`. Ein *token register* enthält eine Reihe von TEX-Befehlen. Beispielsweise sind `\headline` und `\footline` als *token register* in plain-TEX definiert. Wie sonst auch üblich, gelten die ersten 10 Register für lokale Verwendung als frei. Weitere *token register* können mit dem Befehl `\newtoks` reserviert und benannt werden.

Reservierung	*Zuweisung*	*Ausgabe*
	`\toks0={abc}`	`\the\toks0`
	`\headline={\hss\folio\hss}`	`\the\headline`
	`\toks1=\toks0`	`\the\toks1`
`\newtoks\ToksOne`	`\ToksOne={\vfil}`	`\the\ToksOne`
`\newtoks\ToksTwo`	`\ToksTwo=\ToksOne`	`\the\ToksTwo`

Token register werden intuitiv wie bei einer Zuweisung verwendet, zum Beispiel in `\headline` und `\footline`. Eine Zuweisung hat die syntaktische Form, die beispielsweise in "`\AktuelleNamen={Adam und Eva}`" dargestellt ist.

Die Information wird im *token register* ohne Expandierung gespeichert. Allerdings wird bei der Zuweisung auf die Klammerstruktur geachtet, so daß nur vollständige Gruppen abgespeichert werden können.

Der Inhalt eines *token registers* wird durch ein vorangestelltes "`\the`" referiert. Dies kann bisweilen zu Verwirrung führen, wenn dies vergessen wird, da die nachfolgende Information als eine normale Zuweisung interpretiert wird. Da *token register* eigentlich selten verwendet werden, sollen einige praktische Hilfsroutinen für die Verwendung von *token registern* kurz dargestellt werden.

Nach der Definition von

```
\newtoks\ToksOne   \newtoks\ToksTwo   \newtoks\ToksThree
```

werden die *token register* verändert. Dabei werden die Makros angewendet, die auf der folgenden Seite dargestellt sind:

```
\toks1={one}                \toks4=\FirstOf{\toks1}
\toks2={two}                \toks5=\RestOf{\toks2}
\toks3={{one}{two}}         \toks7=\FirstOf{\toks3}
\ToksOne={\number1}         \toks8=\RestOf\ToksOne
\ToksTwo=\toks2             \toks9=\Union(\toks1,\toks2)
\ToksThree=\toks3           \MoveRest(\toks9 to\toks0)
```

Das Ergebnis ist

```
\the\toks1       →  one          \the\toks4   →  o
\the\toks2       →  two          \the\toks5   →  wo
\the\toks3       →  {one}{two}   \the\toks7   →  one
\the\ToksOne     →  \number1     \the\toks8   →  1
\the\ToksTwo     →  two          \the\toks9   →  onetwo
\the\ToksThree   →  {one}{two}   \the\toks0   →  netwo
```

```
%  1. Vereinigung zweier token register
%
%       Ergebnis   " #1={ <Inhalt von #2> <Inhalt von #3>} "
%
\def\JoinToks#1=(#2+#3){#1=\expandafter\expandafter\expandafter
                        {\expandafter\the\expandafter#2\the#3}}
%=============================================================
%
%  2. Ähnlich, jedoch mit der Angabe eines Ziels
%     Ergebnis   " { <Inhalt von #1> <Inhalt von #2> } "
%
\def\Union(#1,#2){\expandafter\expandafter\expandafter
                  {\expandafter\the\expandafter#1\the#2}}
%
\def\UpToHere{\relax}%
\def\IgnoreRest#1#2\UpToHere{#1}               % helper macro
\def\IgnoreFirst#1#2\relax\UpToHere{#2}        % helper macro
%=============================================================
%  3. liefert das erste Element eines token register #1
%
\def\First#1{\expandafter\IgnoreRest\the#1{}\UpToHere}
%=============================================================
%  4. liefert das erste Element eines token register #1 mit
%      umgebenden Klammern " { ... } "
%
\def\FirstOf#1{\expandafter\expandafter\expandafter
               {\expandafter\IgnoreRest\the#1{}\UpToHere}}
%=============================================================
%  5.  weist das erste Element eines token register #1
%       auf das zweite token register #2 zu
%
\def\MoveFirst(#1to#2){#2=\FirstOf{#1}}
%=============================================================
%  6.  gibt alle Elemente aus dem token register #1,
%      außer dem ersten aus.
%
\def\Rest#1{\expandafter\IgnoreFirst\the#1\relax\UpToHere}
%=============================================================
%  7.  wie in (6), jedoch mit umgebenden Klammern " { ... }"
%
\def\RestOf#1{\expandafter\expandafter\expandafter
             {\expandafter\IgnoreFirst\the#1\relax\UpToHere}}
%=============================================================
%  8.  weist alle Elemente des token register #1 außer dem
%      ersten auf #2 zu
%
\def\MoveRest(#1to#2){#2=\RestOf{#1}}
```

9 Variationen des Formelsatzes

9.1 Abstandssteuerung

Die folgenden Parameter beschreiben globale Steuerungsmechanismem für den Mathematiksatz. Diese sollte man nicht ohne Not abändern. Sie sind hier der Vollständigkeit halber und um die Leistungsfähigkeit des TEX-Programms zu demonstrieren aufgeführt. Ähnliches ist zu den folgenden Abschnitten dieses Kapitels zu sagen.

Einer dieser Steuerungsparameter ist der Abstand, den eine Formel links und rechts von ihrer Umgebung besitzt: "`\mathsurround`". Dies ist der Platz, der die Formel vom umgebenden Text trennt. Dieser Parameter gibt den zusätzlichen Abstand an. Standardmäßig ist dieser Wert mit Null vorbesetzt. Durch zum Beispiel "`\mathsurround=1pt`" wird der Abstand auf `1pt` gesetzt.

`\thinmuskip`,`\medmuskip` und `\thickmuskip` sind die internen Variablen für *kleinen*, *mittleren* und *großen* Abstand zwischen verschiedenen Formelelementen.

Die Steuerung der Abstände der Formelbestandteile wird im TEX-System aufgrund der Funktion der einzelnen Formelelemente geregelt. Es gibt folgende Elemente:

Funktion	*Befehl*	*Beispiele*
normale Zeichen	`\mathord`	$ABC \ldots xyz \ldots 012 \ldots \alpha\beta \ldots$
(große) Operatoren	`\mathop`	$\sum \int \prod \ldots$
binäre Operatoren	`\mathbin`	$+ - * \ldots$
Relationen	`\mathrel`	$> = < \ldots$
öffnende 'Klammern'	`\mathopen`	$([\{ \ldots$
schließende 'Klammern'	`\mathclose`	$)]\} \ldots$
Satzzeichen	`\mathpunct`	$., \ldots$
Unterformel	`\mathinner`	

Dabei sind in der Spalte 'Befehl' die TEX-Anweisungen angegeben, um zum Beispiel durch `\mathop{\Gamma}` dafür zu sorgen, daß das Zeichen `\Gamma` mit der Funktion 'Operator' gesetzt wird und damit auch das Abstandsverhalten eines Operators erhält. Verpackt man einen solchen Befehl in ein Makro wie etwa in

```
\def\OpGamma{\mathop{\Gamma}},
```

kann durch den Befehl `\OpGamma` die normale Wirkungsweise eines mathematischen Operators mit dem Druckbild eines 'Γ' erreicht werden. Ein Befehl

```
\def\plus{\mathbin{\hbox{\it plus}}}
```

definiert einen binären Operator, der das gleiche Satzverhalten wie ein '+' besitzt. Durch `$17\plus4=21$` wird $17\ plus\ 4 = 21$ erzeugt. "*plus*" arbeitet also wie '+'. Die Wichtigkeit der Unterscheidung der mathematischen Funktionen sieht man zum Beispiel an der Wirkung von `\mid` und `\vert`, die beide das gleiche Bild produzieren, der eine als Relation und der andere als normales Zeichen und der Verwendung von '`<`' und '`>`' als Relation beziehungsweise als Klammern.
Die zweite Formel demonstriert dann die richtige Verwendung.

`$ < x \vert y > = \sum_i x_i y_i$` $< x|y >= \sum_i x_i y_i$

`$\left< x \mid y \right> = \sum_i x_i y_i$` $\langle x \mid y \rangle = \sum_i x_i y_i$

Ein weiteres Beispiel zeigt die Unterschiede bei der Verwendung des Doppelpunktes "`:`". Dieser ist normalerweise als *Relation* definiert. Er wird jedoch auch als *Satzzeichen* mittels `\colon` und als *binärer Operator* verwendet:

```
$$ G \mathbin: H = G - H  $$ % binaerer Operator
$$ G      :    H = G - H  $$ %   gegenueber normal
$$ f  \colon   M  \to N $$ % Satzzeichen
$$ f      :    M  \to N $$ %   gegenueber normal
```

liefert

$$G : H = G - H$$

$$G : H = G - H$$

$$f \colon M \to N$$

$$f : M \to N$$

Die Abstände zwischen den Formelbestandteilen werden durch verschiedene *skip*-Beträge repräsentiert: `\thinmuskip`, `\medmuskip` und `\thickmuskip`. Durch Veränderung dieser Werte wird das gesamte Layout einer Formel verändert. In der folgenden Tabelle sind die Abstandsvorschriften für die einzelnen Typenpaare aufgeführt. Mit "—" markierte Tabellenelemente stehen für "kein Abstand". Entweder ist diese Elementkombination nicht möglich, oder es wird kein Abstand gesetzt. Platzangaben, die in der Tabelle in Klammern "`[ ]`" gesetzt sind, deuten an, daß sie in Index- und Exponentpositionen ignoriert werden.

Zur Verdeutlichung seien auf der übernächsten Seite einige Beispiele in unterschiedlichem Layout gesetzt:

Die Eingabe

```
$$\int {dx \over x^3 \sqrt{\left(x^2+a^2\right)^3} }=
   - {1.0 \over 2 a^2x^2 \sqrt{x^2+a^2} }$$
```

soll mit mehreren Abstandsvariationen dargestellt werden.

rechts / *links*	normale Zeichen	Operatoren	binäre Operatoren	Relationen
normale Zeichen	—	\thin-muskip	[\med-muskip]	[\thick-muskip]
Operatoren	\thin-muskip	\thin-muskip	—	[\thick-muskip]
binäre Operatoren	[\med-muskip]	[\med-muskip]	—	—
Relationen	[\thick-muskip]	[\thick-muskip]	—	—
öffnende ‘Klammern’	—	—	—	—
schließende ‘Klammern’	—	\thin-muskip	[\thick-muskip]	[\thick-muskip]
Satzzeichen	[\thin-muskip]	[\thin-muskip]	[\thick-muskip]	[\thick-muskip]
Unterformel	[\thin-muskip]	\thin-muskip	[\thick-muskip]	[\thick-muskip]

rechts / *links*	öffnende ‘Klammern’	schließende ‘Klammern’	Satzzeichen	Unterformel
normale Zeichen	—	—	—	[\thin-muskip]
Operatoren	—	—	—	[\thin-muskip]
binäre Operatoren	[\med-muskip]	—	—	\med-muskip
Relationen	[\thick-muskip]	—	—	[\thick-muskip]
öffnende ‘Klammern’	—	—	—	—
schließende ‘Klammern’	—	—	—	[\thin-muskip]
Satzzeichen	[\thin-muskip]	[\thin-muskip]	[\thin-muskip]	[\thin-muskip]
Unterformel	[\thin-muskip]	—	[\thin-muskip]	[\thin-muskip]

Abstände zwischen Formelelementen

Ausgabe mit	`\thinmuskip`
	`\medmuskip`
	`\thickmuskip`

$\int \frac{dx}{x^3\sqrt{(x^2+a^2)^3}} = -\frac{1.0}{2a^2x^2\sqrt{x^2+a^2}}$	0.5-fache des des Normalwertes
$\int \frac{dx}{x^3\sqrt{(x^2+a^2)^3}} = -\frac{1.0}{2a^2x^2\sqrt{x^2+a^2}}$	normal
$\int \frac{dx}{x^3\sqrt{(x^2\ +\ a^2)^3}} = -\frac{1.0}{2a^2x^2\sqrt{x^2\ +\ a^2}}$	2.0-fache des des Normalwertes
$\int \frac{dx}{x^3\sqrt{(x^2\ +\ a^2)^3}} = -\frac{1.0}{2a^2x^2\sqrt{x^2\ +\ a^2}}$	3.0-fache des des Normalwertes

9.2 Eigene mathematische Symbole

Gelegentlich wünscht sich der Autor eigene mathematische Symbole, die er sich durch Zusammensetzung aus anderen Zeichen erzeugt. Im folgenden sei dies beispielhaft dargestellt:

```
\def\kasten#1{\mathop{\mkern0.5\thinmuskip
                      \vbox{\hrule
                            \hbox{\vrule
                                  \hskip#1
                                  \vrule height#1 width 0pt
                                  \vrule}%
                            \hrule}%
                      \mkern0.5\thinmuskip}}
```

definiert ein Makro `\kasten`, das ein Quadrat mit angebbarer Größe ausgibt. Gleichzeitig wird das Ergebnis als *(großer) Operator* interpretiert. Durch den Aufruf

```
$$\kasten{7pt}_{i=1}^n\Gamma_i = B$$
```

erhält man

$$\prod_{i=1}^{n} \Gamma_i = B$$

Wenn eine solche Eigenschöpfung sich in den unterschiedlichen Satzmodi des Mathematiksatzes auch anständig verhalten soll, muß je nach Satzmodus *display-style, text-style, script-style* oder *scriptscript-style* eine unterschiedliche Ausgabe erreicht werden. Dies bewirkt der Befehl `\mathchoice`, der für jeden Satzmodus die zu wählenden Ausgabebefehle angibt. Durch die endgültige Fassung

```
\def\Kasten{\mathchoice{\kasten{8pt}}%        display-style
                        {\kasten{6pt}}%       text-style
                        {\kasten{4pt}}%       script-style
                        {\kasten{3pt}}}%      scriptscript-style
```

wird ein Makro `\Kasten` definiert, das dies erbringt.

```
$$                    \Kasten_{i=1}^n \Gamma_i=B$$
$$\textstyle          \Kasten_{i=1}^n \Gamma_i=B$$
$$\scriptstyle        \Kasten_{i=1}^n \Gamma_i=B$$
$$\scriptscriptstyle \Kasten_{i=1}^n \Gamma_i=B$$
```

liefern

$$\square_{i=1}^{n}\Gamma_i = B$$

$$\square_{i=1}^{n}\Gamma_i = B$$

$$\square_{i=1}^{n}\Gamma_i=B$$

$$\square_{i=1}^{n}\Gamma_i=B$$

Durch `\def\op{\mathbin{\kasten{5pt}}` kann das gleiche Basismakro auch zur Definition eines binären Operators verwendet werden (hier in der vereinfachten Form ohne `\mathchoice`).

```
$$\def\op{\mathbin{\kasten{3.5pt}}}
  f_1 \op f_2 \op f_3 \op \ldots \op f_n = \Kasten_{i=1}^n f_i $$
```

liefert

$$f_1 \square f_2 \square f_3 \square \ldots \square f_n = \square_{i=1}^{n} f_i$$

Eine Verwendung eines vergrößerten Zeichens aus dem Symbolfont liefert das folgende Beispiel:

```
\font\bigmath=cmsy10 scaled \magstep4  % fuer das grosse Zeichen
\def\Stern{\mathop{\vphantom{\sum}%
                   \lower2.5pt\hbox{\bigmath\char3}}}
$$\Stern_{i=1}^n \Gamma_i=C$$
```

ergibt

$$\mathop{\ast}_{i=1}^{n} \Gamma_i = C$$

Bei der Verwendung im Textsatz, also `$\Stern_{i=1}^n \Gamma_i=C$`, erhält man das Ergebnis $\ast_{i=1}^{n} \Gamma_i = C$.

9.3 Layout-Veränderungen

Das mathematische Layout ist eigentlich recht standardisiert. Einige Abwandlungen finden sich jedoch bei der Darstellung hervorgehobener Formeln. Eine davon ist, daß mathematische Formeln linksbündig und nicht zentriert gesetzt werden. Der zentrierte Satz mathematischer Formeln ist in TEX jedoch eine ziemlich interne Geschichte. Dennoch läßt sich das andere Verhalten erreichen.

Zunächst sei auf die beiden Befehle `\everymath` und `\everydisplay` hingewiesen. Beide Befehle haben nur einen Parameter, und zwar eine beliebige Folge von Befehlen. Diese werden gespeichert und dann automatisch zu Beginn einer Formel im Text (bei `\everymath`) oder zu Beginn einer hervorgehobenen Formel (direkt nach `$$`) ausgeführt.† Die folgenden Befehle stellen nun das Mathematiklayout um:

```
\newdimen\mathindent       % Betrag, um den eingerueckt wird
\mathindent=\parindent     % Vorbesetzung (=Absatzeinzug)
\def\eqno{$\hfill$}
\def\leqno{$\hfill$}
\long\def\leftdisplay#1$${\line{\hskip\mathindent
                              $\displaystyle#1$\hfil}$$}
\everydisplay{\leftdisplay}
\catcode`\@=11      % Voruebergehend ist '@' ein Buchstabe
                    % Es werden interne TeX-Befehl verwendet.
\def\eqalignno#1{%
        \displ@y \tabskip=0pt
        \advance\displaywidth by -\mathindent
        \vbox{%
           \halign to \displaywidth{%
               \hfil$\displaystyle{##}$\tabskip=0pt
              &$\displaystyle{{}##}$\hfil\tabskip=\centering
               &\llap{$###$}\tabskip=0pt\crcr#1\crcr}}}
\def\leqalignno{\eqalignno}
\catcode`\@=12      % '@' ist wieder wie vorher
```

Bei der Definition einiger Befehle ist die recht intime Kenntnis der Standardmakros Voraussetzung. Hier soll jetzt aber nur die Leistungsfähigkeit des TEX-Systems demonstriert werden. Dennoch seien einige Definitionen kurz erläutert:
Durch den Befehl `\everydisplay{\leftdisplay}` wird *automatisch* nach einem `$$` der Befehl `\leftdisplay` eingesetzt. Dieser Befehl hat als Begrenzung für seinen Parameter die Zeichenfolge `$$`. Damit geht der ganze Inhalt der Formeleingabe als Parameter an `\leftdisplay`. Dort wird die Eingabe normal gesetzt, nur anschließend anders ausgegeben.

---

† Es gibt noch einige andere `\every...`-Befehle, mit denen Befehlsfolgen zur späteren automatischen wiederholten Ausführung gespeichert werden können:
`\everypar` für Befehle zu Beginn eines Absatzes
`\everyhbox` für Befehle zu Beginn jeder `\hbox`
`\everyvbox` für Befehle zu Beginn jeder `\vbox`

Die internen Makros `\eqno`, `\leqno`, `\eqalignno` und `\leqalignno` wurden umdefiniert, damit sie hierzu noch passen. Insbesondere wird `\eqalignno` neu gestaltet. Dies ist nötig, da dort intern mit `\halign` gearbeitet wird. Alle Befehle zur linksseitigen Numerierung werden auf rechtsseitige Numerierung abgebildet, da links im Normalfall jetzt ja kein Platz ist.
Jetzt sei an einigen Beispielen noch die Funktionsfähigkeit demonstriert:

```
$$ A + B = C $$
$$ A + B = C \eqno (2) $$
$$ A + B = C \leqno (3) $$
$$ \eqalign{ A + B &= C \cr
               C   &= A + B \cr}$$
$$ \eqalignno{ A + B &= C     & (6) \cr
                   B &= C - A &     \cr
                   A &= C - B & (8) \cr}$$
$$ \det A= \left\vert \matrix { a & b & c \cr
                                b & c & a \cr
                                c & a & b \cr}
          \right\vert \leqno (X) $$
```

liefern die folgende Ausgabe, wobei die Größe des linken Einzugs durch `\mathindent` — hier gleich `\parindent` — bestimmt ist.

$$A + B = C$$

$$A + B = C \eqno (2)$$

$$A + B = C \leqno (3)$$

$$\begin{aligned} A + B &= C \\ C &= A + B \end{aligned}$$

$$\begin{aligned} A + B &= C & (6) \\ B &= C - A \\ A &= C - B & (8) \end{aligned}$$

$$\det A = \left\vert \matrix { a & b & c \cr b & c & a \cr c & a & b \cr} \right\vert \leqno (X)$$

9.4 Simulation von Exponenten und Indizes

Die Konvention bei der Eingabe mathematischer Formeln sieht ja den Unterstrich "_" als Kennzeichnung für "*unten*" und das Dach "^" für *oben* vor. Diese Eingabesyntax nun auf eigene Matrix zu übernehmen, ist zunächst keine Schwierigkeit. Es kann ohne weiteres ja

```
\def\myop_#1^#2{...}
```

definiert werden. Dadurch wird diese Eingabeform beim Makroaufruf vorgeschrieben. Sollen die Parameter jedoch auch fehlen dürfen, wird die Sache etwas schwieriger. Für die Definition des Makros, das die folgende Ausgabe liefert, sei auf den Abschnitt 7.7 verwiesen:

`$$ \myop $$`	*liefert*	$\myop$
`$$ \myop_\alpha $$`	*liefert*	$\myop_\alpha$
`$$ \myop^\beta $$`	*liefert*	$\myop^\beta$
`$$ \myop_{aa}^{bb} $$`	*liefert*	$\myop_{aa}^{bb}$

9.5 AMS-Fonts Version 2.0

In einer Reihe von Installationen stehen inzwischen die zusätzlichen mathematischen Symbole der *American Mathematical Society* zur Verfügung. Neben den mathematischen Symbolen beinhalten diese Schriften auch kyrillische Buchstaben, aber das ist nicht Inhalt dieses Buches. Zur Definition der neuen mathematischen Befehle wird meist ein "`\input amssym.def`" *und* "`\input amssym.tex`" notwendig sein, dies ist jedoch durchaus verschieden. Die im folgenden dargestellten Zeichen gehören *nicht* zum Standardzeichensatz von plain-TEX. Sie werden daher hier auch nur tabellarisch aufgeführt.

(Die alten Fassungen der AMS-Fonts, die nur als Pixeldateien ohne Metafont-Quellen verfügbar sind, werden meist durch `\input mssymb` initialisiert.)

"Blackboard" Zeichen

`\Bbb A`	$\mathbb{A}$	`\Bbb N`	$\mathbb{N}$
`\Bbb B`	$\mathbb{B}$	`\Bbb O`	$\mathbb{O}$
`\Bbb C`	$\mathbb{C}$	`\Bbb P`	$\mathbb{P}$
`\Bbb D`	$\mathbb{D}$	`\Bbb Q`	$\mathbb{Q}$
`\Bbb E`	$\mathbb{E}$	`\Bbb R`	$\mathbb{R}$
`\Bbb F`	$\mathbb{F}$	`\Bbb S`	$\mathbb{S}$
`\Bbb G`	$\mathbb{G}$	`\Bbb T`	$\mathbb{T}$
`\Bbb H`	$\mathbb{H}$	`\Bbb U`	$\mathbb{U}$
`\Bbb I`	$\mathbb{I}$	`\Bbb V`	$\mathbb{V}$
`\Bbb J`	$\mathbb{J}$	`\Bbb W`	$\mathbb{W}$
`\Bbb K`	$\mathbb{K}$	`\Bbb X`	$\mathbb{X}$
`\Bbb L`	$\mathbb{L}$	`\Bbb Y`	$\mathbb{Y}$
`\Bbb M`	$\mathbb{M}$	`\Bbb Z`	$\mathbb{Z}$

Normale Zeichen

\square	□	\beth	ב
\blacksquare	■	\gimel	ג
\lozenge	◊	\daleth	ד
\blacklozenge	⧫	\digamma	ϝ
\backprime	‵	\varkappa	ϰ
\bigstar	★	\hslash	ℏ
\blacktriangledown	▼	\hbar	ℏ
\blacktriangle	▲	\yen	¥
\vartriangle	△	\checkmark	✓
\triangledown	▽	\circledR	®
\angle	∠	\maltese	✠
\measuredangle	∡	\nexists	∄
\sphericalangle	∢	\Finv	Ⅎ
\circledS	Ⓢ	\Game	⅁
\complement	∁	\Bbbk	𝕜
\varnothing	∅	\eth	ð
\nexists	∄	\diagup	╱
\mho	℧	\diagdown	╲

Binäre Operatoren

\boxdot	⊡	\curlyvee	⋎
\boxplus	⊞	\leftthreetimes	⋋
\boxtimes	⊠	\rightthreetimes	⋌
\centerdot	∙	\dotplus	∔
\boxminus	⊟	\intercal	⊺
\veebar	⊻	\circledcirc	⊚
\barwedge	⊼	\circledast	⊛
\doublebarwedge	⩞	\circleddash	⊝
\Cup	⋓	\divideontimes	⋇
oder \doublecup	⋓	\ltimes	⋉
\Cap	⋒	\rtimes	⋊
oder \doublecap	⋒	\smallsetminus	∖
\curlywedge	⋏		

Relationen

\leqq	$\leqq$	\supseteqq	$\supseteqq$
\geqq	$\geqq$	\Subset	$\Subset$
\leqslant	$\leqslant$	\Supset	$\Supset$
\geqslant	$\geqslant$	\sqsubset	$\sqsubset$
\eqslantless	$\eqslantless$	\sqsupset	$\sqsupset$
\eqslantgtr	$\eqslantgtr$	\preccurlyeq	$\preccurlyeq$
\lesssim	$\lesssim$	\succcurlyeq	$\succcurlyeq$
\gtrsim	$\gtrsim$	\curlyeqprec	$\curlyeqprec$
\lessapprox	$\lessapprox$	\curlyeqsucc	$\curlyeqsucc$
\gtrapprox	$\gtrapprox$	\precsim	$\precsim$
\approxeq	$\approxeq$	\succsim	$\succsim$
\lessdot	$\lessdot$	\precapprox	$\precapprox$
\gtrdot	$\gtrdot$	\succapprox	$\succapprox$
\lll	$\lll$	\vartriangleleft	$\vartriangleleft$
\llless	$\llless$	\vartriangleright	$\vartriangleright$
\ggg	$\ggg$	\trianglelefteq	$\trianglelefteq$
\gggtr	$\gggtr$	\trianglerighteq	$\trianglerighteq$
\lessgtr	$\lessgtr$	\vDash	$\vDash$
\gtrless	$\gtrless$	\Vdash	$\Vdash$
\lesseqgtr	$\lesseqgtr$	\Vvdash	$\Vvdash$
\gtreqless	$\gtreqless$	\smallsmile	$\smallsmile$
\lesseqqgtr	$\lesseqqgtr$	\shortmid	$\shortmid$
\gtreqqless	$\gtreqqless$	\smallfrown	$\smallfrown$
\doteqdot	$\doteqdot$	\shortparallel	$\shortparallel$
\Doteq	$\Doteq$	\bumpeq	$\bumpeq$
\eqcirc	$\eqcirc$	\between	$\between$
\risingdotseq	$\risingdotseq$	\Bumpeq	$\Bumpeq$
\circeq	$\circeq$	\pitchfork	$\pitchfork$
\fallingdotseq	$\fallingdotseq$	\varpropto	$\varpropto$
\triangleq	$\triangleq$	\backepsilon	$\backepsilon$
\backsim	$\backsim$	\blacktriangleleft	$\blacktriangleleft$
\thicksim	$\thicksim$	\blacktriangleright	$\blacktriangleright$
\backsimeq	$\backsimeq$	\therefore	$\therefore$
\thickapprox	$\thickapprox$	\because	$\because$
\subseteqq	$\subseteqq$		

Verneinte Relationen

\nless	$\nless$	\nsucc	$\nsucc$
\ngtr	$\ngtr$	\npreceq	$\npreceq$
\nleq	$\nleq$	\nsucceq	$\nsucceq$
\ngeq	$\ngeq$	\precneqq	$\precneqq$
\nleqslant	$\nleqslant$	\succneqq	$\succneqq$
\ngeqslant	$\ngeqslant$	\precnsim	$\precnsim$
\nleqq	$\nleqq$	\succnsim	$\succnsim$
\ngeqq	$\ngeqq$	\precnapprox	$\precnapprox$
\lneq	$\lneq$	\succnapprox	$\succnapprox$
\gneq	$\gneq$	\nsim	$\nsim$
\lneqq	$\lneqq$	\ncong	$\ncong$
\gneqq	$\gneqq$	\nshortmid	$\nshortmid$
\lvertneqq	$\lvertneqq$	\nshortparallel	$\nshortparallel$
\gvertneqq	$\gvertneqq$	\nmid	$\nmid$
\lnsim	$\lnsim$	\nparallel	$\nparallel$
\gnsim	$\gnsim$	\nvdash	$\nvdash$
\lnapprox	$\lnapprox$	\nvDash	$\nvDash$
\gnapprox	$\gnapprox$	\nVdash	$\nVdash$
\nprec	$\nprec$	\nVDash	$\nVDash$
\ntriangleleft	$\ntriangleleft$	\ntrianglelefteq	$\ntrianglelefteq$
\ntriangleright	$\ntriangleright$	\ntrianglerighteq	$\ntrianglerighteq$
\nsubseteq	$\nsubseteq$	\varsubsetneq	$\varsubsetneq$
\nsupseteq	$\nsupseteq$	\varsupsetneq	$\varsupsetneq$
\nsubseteqq	$\nsubseteqq$	\subsetneqq	$\subsetneqq$
\nsupseteqq	$\nsupseteqq$	\supsetneqq	$\supsetneqq$
\subsetneq	$\subsetneq$	\varsubsetneqq	$\varsubsetneqq$
\supsetneq	$\supsetneq$	\varsupsetneqq	$\varsupsetneqq$

Pfeile

\leftleftarrows	⇇	\circlearrowleft	↺
\rightrightarrows	⇉	\circlearrowright	↻
\leftrightarrows	⇆	\Lsh	↰
\rightleftarrows	⇄	\Rsh	↱
\Lleftarrow	⇚	\upuparrows	⇈
\Rrightarrow	⇛	\downdownarrows	⇊
\twoheadleftarrow	↞	\upharpoonleft	↿
\twoheadrightarrow	↠	\upharpoonright	↾
\leftarrowtail	↢	\restriction	↾
\rightarrowtail	↣	\downharpoonleft	⇃
\looparrowleft	↫	\downharpoonright	⇂
\looparrowright	↬	\multimap	⊸
\leftrightharpoons	⇋	\rightsquigarrow	⇝
\rightleftharpoons	⇌	\leftrightsquigarrow	↭
\curvearrowleft	↶	\leftarrow	←
\curvearrowright	↷		
\nrightarrow	↛	\nleftrightarrow	↮
\nLeftarrow	⇍	\nLeftrightarrow	⇎
\nRightarrow	⇏		

Öffnende Klammern

\ulcorner	⌜
\llcorner	⌞

Schließende Klammern

\urcorner	⌝
\lrcorner	⌟

10 Fehlermeldungen

10.1 Format der Fehlermeldungen

Die Eingabedatei `test1.tex`, mit nur den beiden Zeilen

```
\def\Spruch{Wer arbeit, macht Fehler.{\fb Vermeydet Fehler!}}
Es gilt: \Spruch, oder doch nicht?
```

als Inhalt, erzeugt folgendes Protokoll mit einer Fehlermeldung:

```
(test1.tex
! Undefined control sequence.
\Spruch ->Wer arbeit, macht Fehler.{\fb
                                        Vermeydet Fehler!}
l.2 Es gilt: \Spruch
                    , oder doch nicht?
?
```

Die Protokollzeile, die mit einem Ausrufezeichen* beginnt, zeigt die Fehlermeldung. Das letzte Element in der folgenden Zeile hat den Fehler produziert. Hier ist dies der unbekannte Befehl "`\fb`". Die nachfolgende Zeile enthält die folgende, noch nicht bearbeitete Information. Die Zeile mit "`l.2 Es gilt: ...`" zeigt, daß der Fehler durch die Eingabezeile 2 mit dem Makroaufruf `\Spruch` verursacht wurde.

Durch die Eingabe von "`h`" erhält man die folgende zusätzliche Information:

```
The control sequence at the end of the top line
of your error message was never \def'ed. If you have
misspelled it (e.g., `\hobx'), type `I' and the correct
spelling (e.g., `I\hbox'). Otherwise just continue,
and I'll forget about whatever was undefined.
```

In diesem Fall stimmt die ausgegebene Vermutung, eigentlich sollte in der Eingabe ein `\bf` stehen.

---

* Übrigens kann so auch automatisch in Batchläufen ein TEX-Protokoll auf Fehlermeldungen untersucht werden: Alle Zeilen eines log-files, die mit einem Ausrufezeichen beginnen, enthalten Fehlermeldungen.

10.2 Verhaltensweisen bei aufgetretenen Fehlern

Keine TEX-Eingabe ist von vornherein perfekt, es wird in fast allen Fällen zu Fehlern und damit zu Fehlermeldungen des TEX-Programms kommen.

Im Dialog mit dem Programm stehen dem Anwender einige einfache Korrekturmöglichkeiten zur Verfügung. Allerdings kann das üblicherweise vorliegende Eingabefile nicht direkt während des TEX-Laufes korrigiert werden.

Im Fehlerfall wird eine Fehlermeldung erzeugt. Nach der Eingabe von '`?`' erhält man die Aktionsmöglichkeiten, wie im nachfolgenden Beispiel, aufgezählt. Die Eingabe '`H`' liefert eine nähere Fehlererläuterung, wie oben in dem Beispiel demonstriert.

```
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.
```

Die wichtigsten Aktionen seien näher erläutert:

(return) Eine leere Eingabe *(return)* läßt das TEX-Programm bis zur nächsten Unterbrechung — Eingabeende oder neuer Fehler — einfach weiterlaufen.

`I` Der auf das Zeichen "`I`" folgende Text, wird an der Unterbrechungsposition eingefügt. So kann in obigem Beispiel durch die Eingabe "`I\bf`", der Befehl `\bf` zusätzlich eingefügt werden. Diese Einfügung wird allerdings nicht in die Eingabedatei geschrieben.

`X E` Die Eingabe `X` oder `E` bricht das Programm *sofort* ab. Einige Implementierungen haben die schöne Eigenschaft, nach einer Eingabe von '`E`' direkt ein Editorprogramm zu starten und an die richtige Zeile zu positionieren.

`S` Die Eingabe `S` läßt das Programm weiter laufen, ohne bei neuen Fehlern anzuhalten, allerdings werden die Fehler auch auf dem Bildschirm protokolliert. (Dies ist auch durch den Befehl `\scrollmode` erreichbar.)

`R Q` Bei diesen Eingaben werden Fehler nur noch in die Protokolldatei geschrieben. Bei der Eingabe von '`R`' wird auch bei schweren Fehlern (fehlende Eingabedateien) nicht mehr angefragt. Dies entspricht den `\nonstopmode`. Durch '`Q`' wird wie durch den Befehl `\batchmode` bei einigen Fehlern, zum Beispiel nicht gefundene Eingabedateien, noch angehalten.

Durch die anderen Befehle kann man in eingeschränkter Weise versuchen, um den Fehler herumzukommen. Dies ist oft sehr schwierig. Normalerweise wird ein neuer Versuch nach einer Bearbeitung der Eingabedatei vorgezogen.

10.3 Häufige Fehler und ihre Ursachen

Die folgenden Beispiele enthalten eine Aufstellung der in der Praxis am häufigsten auftretenden Fehler. Einige Fehler sind fast nur als *Folgefehler* zu erreichen. Dies kann schon mal bei der Fehlersuche zu gewissen Irritationen führen.

```
Undefined control sequence
The control sequence at the end of the top line
of your error message was never \def'ed. If you have
misspelled it (e.g., `\hobx'), type `I' and the correct
spelling (e.g., `I\hbox'). Otherwise just continue,
and I'll forget about whatever was undefined.
```

Typischerweise wurde ein Befehl verwendet, den es nicht gibt. Sehr häufig ist das Vergessen eines Leerzeichens am Befehlsende, zum Beispiel im Wort `anschlie\ssend`. Der *gemeinte* Befehl "`\ss`" wird nun nicht erkannt, statt dessen aber "`\ssend`", und diesen Befehl gibt es nicht.
Hier kann durch "`I\ss end`" der Schreibfehler korrigiert werden.

```
Missing { inserted
```

mit dem zusätzlichen Hilfstext:

```
A left brace was mandatory here, so I've put one in.
You might want to delete and/or insert some corrections
so that I will find a matching right brace soon.
(If you're confused by all this, try typing `I}' now.)
```

TEX hat eine — seiner Meinung nach — fehlende öffnende Klammer eingesetzt. Dies geschieht etwa bei der Eingabe "`\hbox Kiste`". Die Klammerstruktur in der Eingabe ist fehlerhaft. Typischerweise führt dies später zu Folgefehlern.

```
I've run across a `}' that doesn't seem to match anything.
For example, `\def\a#1{...}' and `\a}' would produce
this error. If you simply proceed now, the `\par' that
I've just inserted will cause me to report a runaway
argument that might be the root of the problem. But if
your `}' was spurious, just type `2' and it will go away.
```

Eine schließende Klammer wird als Endesymbol betrachtet, aber es gibt keine dazugehörende öffnende Klammer.

```
Missing number, treated as zero
```

TEX erwartete an dieser Stelle eine Zahl und hat sie nicht bekommen. Zum Beispiel `\pageno=\par` führt zu dieser Situation.

```
Use of ... doesn't match its definition
If you say, e.g., `\def\a1{...}', then you must always
put `1' after `\a', since control sequence names are
made up of letters only. The macro here has not been
followed by the required stuff, so I'm ignoring it.
```

Ein Makro wurde benutzt, das nur in bestimmter textueller Umgebung auftreten darf. Der Aufruf dieses Makros ist zu korrigieren.

```
Missing  ... inserted
```

Es wurde ein Korrekturversuch durch Einfügen dieses Elements gemacht. Dies kann etwa in "`missing } inserted`" bei einer Eingabe von "`\hbox{\smallskip`" geschehen.

```
Illegal unit of measure (pt inserted).
```

oder

```
Dimensions can be in units of em, ex, in, pt, pc,
cm, mm, dd, cc, bp, or sp; but yours is a new one!
I'll assume that you meant to say pt, for printer's points.
To recover gracefully from this error, it's best to
delete the erroneous units; e.g., type '2' to delete
two letters. (See Chapter 27 of The TeXbook.)
```

Es wurde eine falsche Dimensionierung angegeben. Zum Beispiel `\vskip 3mc` statt `\vskip 3cm`. TEX hat diese Angabe durch 'pt' ersetzt. Meist führt dies zu falschem Layout.

```
Paragraph ended before ... was complete
```

Die typische Fehlersituation ist hier, daß ein Makroaufruf nicht vollständig ist. Es fehlt eine schließende Klammer. TEX stößt auf das Absatzende, bevor der Aufruf komplett ist.

```
Parameters must be numbered consecutively
```

Die Definition eines eigenen Makros ist falsch, die Parameter sind nicht aufsteigend numeriert, zum Beispiel `\def\mmm#1#3#2{ ... }` statt `\def\mmm#1#2#3{ ... }`.

```
File ended within ...
```

Hier liegt fast immer eine fehlende Klammer zugrunde, die für einen Prozeduraufruf gebraucht wird. Sehr häufig tritt dies in komplizierteren mathematischen Formeln auf. Es wird der Name der Prozedur gemeldet, die nicht zu Ende gegangen ist.

```
I can't find file ...
```

Das bei "`\input`" angegebene *file* existiert nicht.

```
Font ... not loadable: Metric (TFM) file not found.
```

In einem `\font`-Befehl wurde eine nicht existierende Metrikdatei angegeben. Dies ist entweder ein Schreibfehler, oder die dazugehörende Datei ist nicht vorhanden. Diese Fehlermeldung führt etwa dann zu längerem Nachgrübeln, wenn versucht wurde, einen Ziffern enthaltenden Fontnamen zu definieren, etwa in `\font\bold10=cmbx10`. Da beim `\font` Befehl das Gleichheitszeichen auch weggelassen werden darf, wird versucht, eine überhaupt nicht gemeinte Datei "`10=cmbx10`" einzulesen.

```
not loadable: Bad metric (TFM) file
```

oder

```
I wasn't able to read the size data for this font,
so I will ignore the font specification.
[Wizards can fix TFM files using TFtoPL/PLtoTF.]
You might try inserting a different font spec;
e.g., type `I\font<same font id>=<substitute font name>'.
```

Ein defekter Metrikfile wurde gefunden. Das TEX-Programm schlägt vor, einen anderen Font zu definieren.

```
not loaded: Not enough room left
```

Der Speicherplatz hat zum Laden dieses Fonts nicht mehr ausgereicht. Durch die Einstellung "`\tracingstats=1`" erhält man zum Ende des Programmlaufs eine Statistik über den verwendeten Speicherraum.

```
Missing character: There is no ... in font ...
```

Nachdem eine Schrift nicht gefunden wurde, gibt es nach Anwahl dieser nicht existierenden Schrift für jedes Zeichen eine Fehlermeldung. Dies kann aber auch geschehen, wenn Sonderfonts verwendet werden, in denen nicht alle Plätze besetzt sind, so daß hierdurch ein nicht definiertes Zeichen aufgerufen wurde. Diese Fehlermeldung wird nur dann ausgegeben, wenn die Variable `\tracinglostchars` größer Null ist.

```
Underfull \hbox (badness ... ) has occurred ...
overfull \hbox (badness ... ) has occurred ...
Underfull \vbox (badness ... ) has occurred ...
overfull \vbox (badness ... ) has occurred ...
```

Diese Fehlermeldungen sind die häufigsten überhaupt. Meist ist eine Zeile zu lang oder zu kurz, weil das TEX-Programm nicht trennen konnte. Dies ist die typische Ursache für "`overfull \hbox ...`". Der Anfang der entsprechenden Zeile wird protokolliert.

Das Problem mit `\vbox` tritt meist auf, wenn nur `\eject` geschrieben wurde. Dies ist zu korrigieren durch "`\vfill\eject`".

Wurde explizit mit `\vbox` und `\hbox` gearbeitet, wobei bei diesen Dimensionsangaben vollzogen wurden, so fehlt meist ein `\vfill` bzw. ein `\hfill`.

Die typische Situation für eine "`overfull \hbox...`" sind zu lange Zeilen. Es folgt die Meldung "`... pt too wide`". Alle Meldungen mit einer Länge von weniger als 1 pt kann man getrost ignorieren. Solche Längen sind bei der Ausgabe später kaum noch feststellbar. Seit der Version 3 kann durch Setzen des Parameters `\emergencystretch` der Umbruch, insbesondere bei kurzen Satzlängen, günstig beeinflußt werden. Durch den Befehl "`\hfuzz=1pt`" kann übrigens die Protokollierungsgrenze für eine überfüllte `\hbox` angehoben werden.

Die normale Ursache beim Umbruch eines Absatzes ist eine nicht vollständige Trennung. Das TEX-Programm protokolliert jeweils die letzten Wörter in der überfüllten Zeile einschließlich der erlaubten Trennpositionen. So können also Korrekturen vollzogen werden, ohne daß eine explizite Ausgabe auf irgendeinem Ausgabegerät erfolgen muß.

```
Missing # inserted in alignment preamble
There should be exactly one # between &'s, when an
\halign or \valign is being set up. In this case you had
none, so I've put one in; maybe that will work.
```

Der Aufruf eines `\halign` Befehls ist fehlerhaft, und zwar die Musterzeile.

```
Only one # is allowed per tab
```

oder

```
There should be exactly one # between &'s, when an
\halign or \valign is being set up. In this case you had
more than one, so I'm ignoring all but the first.
```

In der Definition einer Musterzeile in `\halign` wurde zweimal ein `#` für dieselbe Spalte gesetzt. Wahrscheinlich fehlt ein `&`.

```
I've inserted something that you may have forgotten.
(See the <inserted text> above.)
```

TEX hat zusätzliche Befehle eingefügt. Vorsicht !

Meist wird ein "`$`" eingefügt, weil das TEX-Programm auf einen Befehl gestoßen ist, der nur im Mathematikmodus erlaubt ist. Dadurch kommt natürlich die nachfolgende Eingabe ziemlich durcheinander, wenn diese keine mathematische Eingabe ist.

```
You've closed more groups than you opened.
Such booboos are generally harmless, so keep going.
```

Es ist eine schließende Klammer "`}`" mehr aufgetreten als vorher öffnende vorhanden waren. Dies sollte der Sauberkeit halber korrigiert werden.

```
Extra }, or forgotten {
```

Es ist eine schließende Klammer "`}`" mehr aufgetreten als vorher öffnende vorhanden waren. Dies sollte der Sauberkeit halber korrigiert werden.

```
I've deleted a group-closing symbol because it seems to be
spurious, as in `$x}$'. But perhaps the } is legitimate and
you forgot something else, as in `\hbox{$x}'. In such cases
the way to recover is to insert both the forgotten and the
deleted material, e.g., by typing `I$}'.
```

Eine Klammer steht an einer syntaktisch unzulässigen Position.

```
Please use \mathaccent for accents in math mode
```

mit dem Zusatztext

```
I'm changing \accent to \mathaccent here; wish me luck.
(Accents are not the same in formulas as they are in text.)
```

Der Fehler wird durch einen Akzentbefehl, wie `\‘` innerhalb des Mathematiksatzes hervorgerufen. Da für den Mathematiksatz grundsätzlich eigene Akzentbefehle verwendet werden, erscheint diese Meldung.

Dies ist fast immer ein Folgefehler: TEX hat ein `$`-Symbol eingefügt, weil es der Meinung war, die Information gehört in den Mathematiksatz. So sind zum Beispiel `^` und `_` als Exponent- und Indexbefehle *nur* in mathematischen Formeln erlaubt. Ähnliches gilt für eine Reihe von Symbolen, die als mathematische Operatoren aufgefaßt werden. Allerdings läuft dann diese zwangseingeschaltete Mathematik fast immer in den normalen Text mit Umlauten hinein!

```
I can't find file ‘daten.tex’.
<*> \input daten
Please type another input file name:
```

TEX kann eine Eingabedatei nicht finden. In diesem Fall wird die Datei `daten.tex` gesucht. Es besteht die Möglichkeit, einen anderen Dateinamen an dieser Stelle anzugeben. Manche Implementierung verlangt an dieser Stelle unbedingt die Eingabe des Namens einer existierenden Datei und fragt immer wieder nach dem Namen einer existierenden Datei, wenn die zuletzt angegebene Datei nicht gefunden wird. In PC-Implentierungen kann man hier zur Not auch `con` eingeben und durch `\endinput` das Eingabeende simulieren. Will man klug vorbauen, so ist es sinnvoll, eine leere Datei mit beispielsweise dem Namen `exit.tex` vorzuhalten.

10.4 Protokollparameter

Einige Kommandos regeln, wie umfangreich das Protokoll der Eingabe erstellt wird. Bei der Fehlersuche wird man häufig eine ausführlichere Version erzeugen. Die wichtigsten Parameter zur Aktivierung der Protokollierung sind:

`\tracingonline=1` Durch diesen Befehl werden die Testausgaben auch auf dem Terminal protokolliert. Im Normalfall werden diese nur in die Protokolldatei geschrieben.

`\tracingcommands=1` Jedes *ausgeführte* Kommando wird protokolliert.

`\tracingcommands=2` Alle Kommandos, auch die in `\if`-Anweisungen übersprungenen, werden protokolliert.

`\tracingmacros=1` Bei Makroaufrufen wird die Besetzung der Parameter protokolliert.

Nach den Eingaben

```
\tracingmacros=1
\def\value#1#2{{\bf #1} $\to$ {\bf #2}}
\value{1\$}{2 DM}
```

erhält man im Protokoll

```
\value #1#2->{\bf #1} $\to $ {\bf #2}
#1<-1\$
#2<-2 DM
\bf ->\fam \bffam \tenbf
\bf ->\fam \bffam \tenbf
```

Ein Hinweis: Wie man sieht, ist der Befehl "`\bf`" selbst wieder ein Makroaufruf.

`\tracingall` *Alle* Protokollierungen werden aktiviert. Die hierdurch erzeugte Information ist sehr umfangreich.

`\tracingstats=1` Am Ende des TEX-Laufes wird eine Statistik über die Ausnutzung des Programmspeichers ausgegeben. Damit kann der noch verfügbare Platz für zusätzlich zu ladende Schriften oder weitere Makros festgestellt werden.

`\tracingstats=2` Eine Teilstatistik wird am Ende jeder erzeugten Ausgabeseite geliefert.

Beispielstatistik (Die Maximalwerte sind implementationsabhängig.)

```
Here is how much of TeX's memory you used:
 564 strings out of 65536
 1668 string characters out of 1894
 13962 words of memory out of 65535
 1059 multiletter control sequences out of 5000
 15679 words of font info for 53 fonts
                           out of 15680 for 255
 0 hyphenation exceptions out of 307
 15i,8n,8p,186b,76s stack positions
                 out of 64i,64n,128p,3000b,640s
```

11 Output-Routinen

11.1 Aufgabe einer Output-Routine

Zunächst hat der normale Benutzer, der nichts am definierten Layout einer Seite ändern möchte, mit einer *Output-Routine* nichts weiter zu schaffen. Erst, wenn er die Struktur einer Ausgabeseite, die ja aus Kopfzeile für die Überschrift, Seitenrumpf und Fußzeile besteht, verändern möchte, muß er eine eigene Output-Routine schreiben. Häufig reicht es allerdings, die Standardroutine zu verändern, um die gewünschte Leistung zu erhalten.

Die Output-Routine ist das Makro beziehungsweise die Befehlsfolge, die vom TEX-Programm aufgerufen wird, wenn eine "Seite voll" ist. "Seite voll" bedeutet, so viel Satzmaterial ist angefallen, daß die Größe von "`\vsize`" erreicht wurde. In diesem Fall wird die Output-Routine aktiviert. Diese kann dann entscheiden, ob die Seite nun tatsächlich mittels "`\shipout`" in die Ausgabedatei gegeben werden soll. Die Standard-Output-Routine vollzieht ihre Aufgabe, indem sie über und unter den Text Kopf- und Fußzeilen montiert. Der Text wird aus eventuellen Einfügungen aus "`\topinsert`", Fußnoten aus "`\footnote`" sowie natürlich dem normalen Eingabetext zusammengesetzt. Um diese Aufgabe zu erfüllen, bekommt die Output-Routine bei ihrer Aktivierung die notwendige Information in bestimmten Registern übergeben:

`\box255` enthält die Seite, das heißt den Text, der gemäß "`\vsize`" umbrochen wurde. Der Inhalt von "`\box255`" ist eine "`\vbox`". In ihr befindet sich die normale Satzinformation *ohne* Fußnoten und Einfügungen durch "`\topinsert`".

`\outputpenalty` ist ein numerisches Register mit der Bewertung der Seitenumbruchstelle. Durch etwa "`\ifnum\outputpenalty>10 ...`" kann dieses abgefragt werden.
Genau betrachtet wird die Output-Routine aufgerufen, wenn die Seite voll ist *oder* wenn eine Umbruchstelle mit

$$\texttt{\textbackslash penalty} \leq -10000,$$

also mehr als 10000 Pluspunkten, erreicht wird.
Der Befehl "`\eject`" etwa erzwingt auf genau diese Weise eine Aktivierung der Output-Routine. Er ist selbst ein Makro und durch

`\def\eject{\par\penalty-10000 }`
definiert. Hier zeigt sich auch eine Methode, an die Output-Routine eine zusätzliche Information über die gewünschte Wirkung mitzugeben, indem man für bestimmte Werte "-10001, -10002 ..." andere Verhaltensweisen ausprogrammiert.

`\insertpenalties` enthält die Zahl der noch nicht abgearbeiteten Einfügungen mittels "`\insert`", etwa durch "`\topinsert`".

`\deadcycles` besitzt als Inhalt die Anzahl der vorherigen Aufrufe der Output-Routine, bei denen kein "`\shipout`" erfolgt, die Information also nur gespeichert wurde. Dies dient der Verhinderung von Endlosschleifen. Der begrenzende Parameter ist "`\maxdeadcycles`".

11.2 Die Standard-Output-Routine

Im folgenden soll die Arbeitsweise der Standardroutine beschrieben werden. Mit ihr erfolgt die Zusammenstellung der Standardseite in plain-TEX. Diese Routine ist definiert durch die folgenden Befehle:

```
\output={\plainoutput}

\def\plainoutput{
        \shipout\vbox {\makeheadline   % 1. Druckseite erstellen
                       \pagebody       %    und ausgeben
                       \makefootline}  %
        \advancepageno                 % 2. Seitenzählung fortschalten
        \ifnum\outputpenalty>-20000    % 3. Sonderbehandlung am
           \else                       %      Programmende
              \dosupereject
           \fi}
```

Das Register "`\output`" speichert die Befehlsfolgen für die Output-Routine. Hier ist nur ein Verweis auf das Makro "`\plainoutput`" eingetragen, welches die eigentliche Aufgabe übernimmt. Dieses besteht aus drei Hauptelementen:

1. Zusammensetzung und Ausgabe der Seite:

   ```
   \shipout\vbox {\makeheadline
                  \pagebody
                  \makefootline}
   ```

   Der Befehl "`\shipout`" gibt die folgende Box in die Ausgabedatei aus. Diese setzt sich zusammen aus:
   a) "`\makeheadline`", Bildung der Kopfzeile
   b) "`\pagebody`", Bildung des Seitenrumpfes
   c) "`\makefootline`", Bildung der Seitenfußzeile
2. In "`\advancepageno`" wird die Seitenzählung fortgeschaltet.
3. Zum Abschluß wird überprüft, ob bei einem Endeaufruf der Outputroutine mit (`\outputpenalty` ≤ -20000) noch nicht abgearbeitete Einfügungen anstehen.

1.a) Erstellung der Kopfzeile

Das Makro "`\makeheadline`" erzeugt eine von den externen Maßen leere "`\vbox`" der Größe 0 pt. Innerhalb dieser Box wird zur Positionierung des Kolumnentitels um eine Leerzeile zurückgegangen, die Kopfzeile ausgegeben und die Box aufgefüllt.

```
\def\makeheadline{\vbox to 0pt
                      {\vskip -22.5pt
                       \line{\vbox to 8.5pt{}\the\headline}
                       \vss}
                  \nointerlineskip}
```

Die merkwürdigen Zahlenangaben beruhen auf der Tatsache, daß eine normale Zeile die Gesamthöhe von 12 pt besitzt, die sich in 8.5 pt und 3.5 pt für Ober- und Unterlängen aufteilt. Wird mit anderen Zeilenabständen gearbeitet, empfiehlt sich hier bestimmt eine Anpassung. Der Schlußbefehl "`\nointerlineskip`" sorgt dafür, daß der Kolumnentitel ohne zusätzlichen Abstand zur folgenden eigentlichen Seiteninformation gesetzt wird.

Durch eine einfache Änderung des "`\vskip-22.5 pt`" zu "`\vskip -34.5pt`" kann zum Beispiel der Kolumnentitel mit zwei Zeilen Abstand vom Text gesetzt werden.

1.b) Erstellung der Informationsseite

Das Makro "`\pagebody`" setzt die eigentliche Textseite. Es ruft noch ein Untermakro "`\pagecontents`" auf, welches die eigentliche Arbeit ausführt.

```
\def\pagebody{\vbox to \vsize{\boxmaxdepth=\maxdepth
                              \pagecontents}}
```

Zunächst wird die maximale "Unterlänge" der Seite auf "`\maxdepth`" beschränkt. Der Wert von `\maxdepth` ist in plain-TEX mit 4 pt besetzt.

```
\def\pagecontents{\ifvoid\topins\else\unvbox\topins\fi
                  \dimen0=\dp255
                  \unvbox255
                  \ifvoid\footins
                  \else
                    \vskip\skip\footins
                    \footnoterule
                    \unvbox\footins
                  \fi
                  \if@raggedbottom\kern-\dimen0\vfil\fi}
```

Diese Seitenkonstruktion sieht etwas komplizierter aus, da hier mehrere Dinge zusammenkommen. Zunächst wird in

```
\ifvoid\topins\else\unvbox\topins\fi
```

geprüft, ob mit "`\topinsert`" eingefügtes Material ausgegeben werden muß. Dies geschieht dann durch "`\unvbox\topins`". Der nächste Befehl "`\dimen0=\dp255`" speichert die Unterlänge der aktuellen Seite für die Verwendung in der `\if`-Abfrage am

Schluß. Durch "`\unvbox255`" wird die Box 255 geleert und ausgegeben. Diese enthält nun endlich die umbrochene Seite. Im Anschluß werden die gespeicherten Fußnoten gesetzt:

```
\ifvoid\footins                    | Fußnoten vorhanden?
   \else                           |
     \vskip\skip\footins           |  etwas Leerraum
     \footnoterule                 |  Trennstrich setzen
     \unvbox\footins               |  Fussnotenbox ausgeben
\fi                                |
```

Zunächst wird geprüft, ob überhaupt welche da sind: "`\ifvoid\footins`". Wenn dies der Fall ist, wird etwas Leerraum vor den Fußnoten gelassen: "`\vskip\skip\footins`", der Trennstrich für Fußnoten erzeugt "`\footnoterule`" sowie und letztendlich die Box ausgegeben, die den gespeicherten Fußnotentext enthält: "`\unvbox\footins`".

Damit ist der Seitenrumpf fertiggestellt.

1.c) Seitenfußzeile

Das Makro "`\makefootline`" ist durch

```
\def\makefootline{\baselineskip=24pt
                  \line{\the\footline}}
```

definiert. Auch hier wird der Untertitel im Abstand einer Leerzeile gesetzt. Der Standardzeilenabstand "`\baselineskip`" wird verdoppelt. Dadurch entsteht der größere Abstand zur Seitenunterschrift.

2. Fortschaltung der Seitenzählung

Eine Aufgabe der Output-Routine ist auch die Fortschaltung der Seitenzählung. Laut Konvention in plain-TEX wird diese, wenn die Seitenzahl größer als Null ist, um 1 erhöht, sonst um 1 erniedrigt.

```
\def\advancepageno{
        \ifnum\pageno < 0 \global\advance\pageno by -1
          \else           \global\advance\pageno by  1 \fi}
```

3. Behandlung beim Programmende

Durch die Befehle

```
\ifnum\outputpenalty>-20000\else
\dosupereject\fi
```

wird eine weitere Seitenausgabe erzwungen, um anstehende Resteinfügungen auch noch auszugeben. "`\dosupereject`" gibt dann eventuell noch eine Leerseite aus.

```
\def\dosupereject{
      \ifnum\insertpenalties>0
        \vbox to -\topskip{\line{}\vss}
        \vfill\supereject\fi}
```

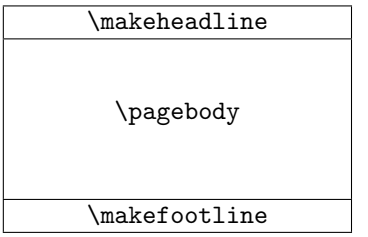
Dies geschieht, indem noch eine Leerseite erzeugt wird. Das folgende "`\supereject`", definiert durch "`\par\penalty-20000`", erzwingt eine nochmalige Vollaktivierung der Output-Routine.

11.3 Variationen der Output-Routine

Die einfachste Anwendung einer Output-Routine ist eigentlich die schlichte Benutzung des `\shipout`-Befehls. Dieser besitzt als Parameter eine nachfolgende Box, welche als Druckseite in die Ausgabedatei geschrieben wird. Angenommen, in einer "`\vbox`" hat man sich eine Seite mit einem speziellen Layout zusammengesetzt. Soll diese Seite nun ohne Kopf- und Fußzeilen ausgegeben werden, kann dies zum Beispiel durch den Befehl `\shipout\box0` bewirkt werden. Hier wird vorausgesetzt, daß die Box 0 die Ausgabeseite enthält.

Auch recht einfach ist die Modifikation der Standardmakros bei der Zusammensetzung der Druckseite. Wir erinnern uns, daß durch die folgenden Befehle die Seite gesetzt wird:

```
\shipout\vbox{\makeheadline
              \pagebody
              \makefootline}
```



Es ist insbesondere leichter, "`\makeheadline`" zu modifizieren, als in dem *token*-Register `\headline` komplizierte Rückwärtspositionierungen einzubauen.
Beispiel:

```
\def\makeheadline{%
      \vbox{\hrule
            \line{%
              \vrule\quad
              $\vcenter{%
                  \vbox{\it\medskip
                        \hbox{Fix \& Fertig}
                        \hbox{Rechenzentrum}
                        \medskip}}$
              \quad\vrule
              \hfil\tenrm\the\headline\hfil
              \vrule\quad
              $\vcenter{%
                  \vbox{\bf\hbox{FFR --- Seite \folio}}}$
              \quad\vrule}%
            \hrule}
      \nointerlineskip}
```

Dabei wird "`\headline`" der Seitentitel zugewiesen.
Nach `\headline={Primzahlberechnung}` erhält man als Seitentitel:

Fix & Fertig *Rechenzentrum*	Primzahlberechnung	**FFR — Seite 175**

Da der Kolumnentitel einer solchen Seite nun wesentlich größer als die normale Seitenüberschrift ist, wird die Gesamtseite nun länger. Es ist daher erforderlich, daß der nachfolgende Seitentext kürzer ausfällt. Demnach muß der Wert von "`\vsize`" reduziert werden.

Mehrspaltige Ausgabe

Zunächst sei daran erinnert, daß der Seitenumbruch nach der Größe "`\vsize`" erfolgt, der Zeilenumbruch nach dem Wert von "`\hsize`". Für die effektive Ausgabe eines zweispaltigen Textes wird wegen des zusätzlichen Leerraums zwischen den beiden Spalten eine effektive Seitenbreite größer $2 \times$ `\hsize` erforderlich sein.

Die Output-Routine wird dann jedesmal aktiviert, wenn *eine Spalte* vollständig gefüllt ist. Ihre Arbeitsweise ist, wie man sich leicht überlegen kann, dann folgendermaßen: Eine linke Spalte einer Seite wird nicht ausgegeben, sondern gespeichert. Erst, wenn auch eine rechte Spalte fertiggestellt ist, wird die Gesamtseite zusammengesetzt und ausgegeben.

Die nötigen Definitionen sehen dann wie folgt aus. Zunächst werden einige notwendige Hilfsregister definiert, mit denen dann in der Output-Routine gearbeitet wird.

```
\newdimen\echteBreite          \newif\ifrechts
\echteBreite = \hsize          \newbox\LinkeSpalte
\hsize=0.45 \hsize
```

Dies legt die Spaltenbreite auf 45 % der Normalbreite fest. Durch "`\newif\ifrechts`" wird eine Abfrage definiert, mit der die Unterscheidung, ob jetzt eine rechte oder linke Spalte kommt, getroffen wird. Orientiert an der Standardroutine definieren wir:

```
\def\doppelSpalte{%
   \ifrechts
      \shipout\vbox{\longheadline
                    \hbox to \echteBreite
                        {\box\LinkeSpalte
                         \hfil
                         \leftline{\pagebody}}
                         \longfootline}
      \global\rechtsfalse
      \advancepageno
   \else
      \global\setbox\LinkeSpalte=
               \leftline{\pagebody}
      \global\rechtstrue
   \fi
   \ifnum\outputpenalty>-20000\else
       \dosupereject \fi}
\output={\doppelSpalte}
```

Die Befehle `\longheadline` und `\longfootline` sind noch nicht definiert worden. Diese wären als Abwandlung von `\makeheadline` und `\makefootline` zu gestalten. Dabei ist der `\line`-Befehl, welcher nichts weiter als `\hbox to \hsize` heißt, zu ersetzen durch ein `\hbox to \echteBreite`. Damit wäre eine dem normalen Layout angepaßte Gestaltung vollzogen.

Dies ist allerdings eine einfache Fassung der Problemstellung, da Fußnoten und Einfügungen noch spezifisch für jede Spalte abgewickelt werden. Bei der Anwendung ist es nun allerdings nicht sicher, daß der Text auch jedesmal auf einer rechten Spalte endet. Daher sollte am Ende der Eingabe noch eine Auffüllung der rechten Spalte stehen:

```
\ifrechts\hbox{}\vfill\eject\fi
```

Weiterhin wird häufig gewünscht, den Text der beiden letzten Spalten nicht in eine lange und eine kurze, sondern in zwei gleich lange Spalten aufzuteilen. Diese Operation ist etwas schwieriger. Hier soll nur ein Rezept dazu angegeben werden: Die beiden Spalten werden zunächst mit "Entbättern" durch `\unvbox` in einer neuen `\vbox` vereinigt. Zu dieser `\vbox` kann die Höhe mittels `\ht` referiert werden, die dann als Berechnungsgrundlage für eine `\vsplit`-Abspaltoperation verwendet wird. Das Verfahren ist im Abschnitt "Box-Teilausgaben" exemplarisch vorgeführt.

11.4 Seitenspezifische Textmarkierungen

Eng mit der Output-Routine verbunden sind Markierungen im Text, die später seitenspezifisch ausgewertet werden. Dies sind etwa Abschnittsüberschriften, an denen man hinterher, wenn die Seite fertig umbrochen ist, feststellen möchte, welches der erste und welches der letzte Eintrag auf der aktuellen Seite ist. Eine Anwendung ist etwa die Übernahme in den Kolumnentitel. Um etwa ein Seitenlayout der folgenden Form zu erzeugen, bei der die Bezeichnungen des ersten und letzten Abschnitts der aktuellen Seite jeweils links und rechts im Titel gesetzt werden, sind die nachfolgenden Befehle notwendig.

Didone 177 *Mécane*

angesetzt. Die Haar- und Grundstriche der Didone unterscheiden sich kräftig. Die Rundungsachse steht senkrecht.

Französische Renaissance-Antiqua — Sie weist etwas größere Unterschiede in der Strichdicke auf, der Querstrich des 'e' liegt waagerecht. *Garalde*

Serifenlose Linear-Antiqua — Diese Schriften werden häufig auch als "Grotesk" bezeichnet. Sie besitzen keine Serifen mehr. *Lineale*

Serifenbetonte Linear Antiqua — Die Haar- und Grundstriche unterscheiden sich kaum. *Mécane*

Der Befehl `\mark{..information..}` trägt jeweils die zu speichernde Information ein. In unserem Beispiel sind dies die Befehle:

```
...
\mark{Didone}
\mark{Garalde}
\mark{Lineale}
\mark{M\'ecane}
...
```

Diese Kommandos werden jeweils zu Beginn des betreffenden Absatzes gegeben. Innerhalb der Output-Routine kann die gespeicherte Information über die folgenden Anweisungen referiert werden.

`\botmark` ist der Text, der *zuletzt* mit `\mark` bis zum Ende der aktuellen Seite gespeichert wurde.

`\firstmark` liefert — falls auf der aktuellen Seite `\mark`-Befehle erfolgten — den *ersten* Eintrag von diesen zurück, sonst die Information aus der letzten vorangehenden `\mark`-Anweisung.

`\topmark` gibt den zu Beginn der aktuellen Seite gültigen `\mark`-Text aus. Das heißt, `\topmark` ist gleich dem `\botmark` der Vorgängerseite.

Die Wirkungsweise der Befehle sei noch an der folgenden Tabelle veranschaulicht:

Seite	gegebene `\mark`-Befehle	Ergebnis bei `\topmark`	Ergebnis bei `\firstmark`	Ergebnis bei `\botmark`
1	—	*Leertext*	*Leertext*	*Leertext*
2	`\mark{A}`	*Leertext*	`A`	`A`
3	—	`A`	`A`	`A`
4	`\mark{B}` + `\mark{C}`	`A`	`B`	`C`
5	`\mark{D}`	`C`	`D`	`D`
6	—	`D`	`D`	`D`

In dem Beispiel wird der zugehörige Kolumnentitel also definiert durch

```
\headline={\rlap{\centerline{\tenrm\folio}}% Seitennummer
          \tenit \topmark \hss \botmark}
\footline={\hss}
```

Die Konstruktion "`\rlap{\centerline{\tenrm\folio}}`" bewirkt eine Zentrierung der Seitennummer bezüglich der gesamten Zeilenbreite, die unabhängig von eventuellem linkem oder rechtem Text ist. Eine Angabe der Form

```
\line{\tenit\topmark\hss{\tenrm\folio}\hss\botmark}
```

sorgt nur für eine Zentrierung bezüglich des mittleren Leerraums.

Noch eine Anmerkung:

Bei Lexika-ähnlichen Texten ist es nun auch eine Geschmacksache, ob nicht statt des Titels des angefangenen Abschnitts die Bezeichnung des ersten neuen Abschnitts ausgegeben werden soll. In diesem Fall ist im Beispiel "`\topmark`" durch "`\firstmark`" zu ersetzen.

12 Anwendungsbeispiele

In diesem Kapitel sollen beispielhaft einige umfangreichere Anwendungen in verschiedenen Fällen dargestellt werden. Naturgemäß werden eine Reihe von Makros definiert, wobei die Makros in den verschiedenen Beispielen nicht unbedingt alle miteinander verträglich sein müssen. Dies gilt leider für viele TEX-Anwendungen.

12.1 Balkendiagramme

Eine graphische Anwendung ist die Benutzung der `\vrule` und `\hrule`-Befehle zur Erzeugung von Balkendiagrammen. Im folgenden Beispiel wird dies exemplarisch in einem `\diagramm`-Makro vorgeführt:

Besetzung der Makroparameter von `\diagramm`

`#1` Längeneinheit zum Beispiel: 'cm' oder 'pt' oder ein mit "`\newdimen`" erzeugtes Längenregister

`#2` Breite des Diagramms (in Einheiten von `#1`)

`#3` Höhe des Diagramms (in Einheiten von `#1`)

`#4` Überschrift (wird in der Mitte oben gesetzt)

`#5` Unterschrift (wird in der Mitte unten gesetzt)

`#6` Nutzinformation (von der Struktur einer "`\hbox`")

Innerhalb von `#6` sind folgende Befehle möglich:

`\bbalken{` *Höhe* `}{` *Untertitel* `}` liefert jeweils einen fetten Balken ('bold'). Die Information aus "*Untertitel*" wird zentriert gesetzt.

`\sbalken{` *Höhe* `}{` *Untertitel* `}` liefert einen senkrecht schraffierten Balken.

`\hbalken{` *Höhe* `}{` *Untertitel* `}` liefert einen 'hohlen' Balken.

`\bbalkentop{` *Höhe* `}` setzt ein Balken oben auf den vorangehenden,
`\sbalkentop{` *Höhe* `}` entsprechend schraffiert,
`\hbalkentop{` *Höhe* `}` entsprechend hohl.

`\neubreite{` *Wert* `}` setzt die Balkenbreite in Einheiten von `#1` neu fest.

Die Untermakros werden lokal beim Aufruf von "`\diagramm`" stets neu definiert.

Die umfangreichen Makros dazu sehen so aus:

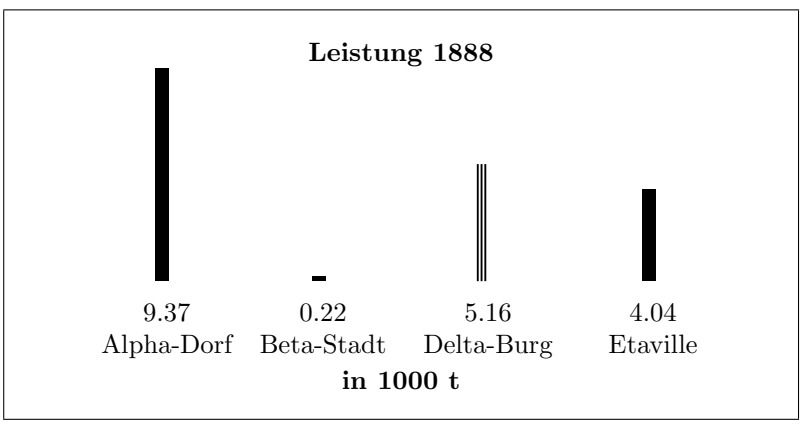
```
% Hilfsmakro fuer eine Einrahmung
\def\Einrahmung#1{\vbox{\hrule
                        \hbox{\vrule\vbox{#1}\vrule}%
                        \hrule}}


\def\diagramm#1#2#3#4#5#6{{%
%
%  Konstanten
%
   \dimen1=0.3cm % Breite eines Balkens (Vorbesetzung)
%
%  Breite eines Balkens neu setzen
%
   \def\neubreite##1{\dimen1=##1#1\relax}%
%
%  1.  Untere Beschriftung eines Balkens setzen (Hilfsmakro)
%
\def\markier##1{\setbox0=\hbox{##1}%
   \hbox to 0pt{\kern0.5\dimen1%
   \kern-0.5\wd0\vbox to 0pt{\kern0.3cm\box0\vss}\hss}}%
%
%   2.a Makro für einen fetten Balken
%
\def\bbalken##1##2{\markier{##2}\dimen3=##1#1%
   \vrule width\dimen1 height##1#1}%
%
%  2.b  Makro für einen fetten Balken, der auf den
%       vorhergehenden gesetzt wird
%
\def\bbalkentop##1{\kern-\dimen1{%
   \dimen0=##1#1\advance\dimen0 by \dimen3%
   \vrule width\dimen1 height\dimen0 depth-\dimen3}%
   \advance\dimen3 by ##1#1}%
%
%   3.a Makro für einen hohlen Balken
%
\def\hbalken##1##2{\markier{##2}%
   \dimen3=##1#1%
   {\dimen2=\dimen1\advance\dimen2 by-0.1cm
   \vrule height##1#1 width0.05cm
   \vrule height0.05cm width\dimen2  \kern-\dimen2
   \dimen0=-0.05cm \advance\dimen0 by ##1#1\relax
   \vrule height##1#1 width\dimen2 depth-\dimen0
   \vrule height##1#1 width0.05cm}}
```

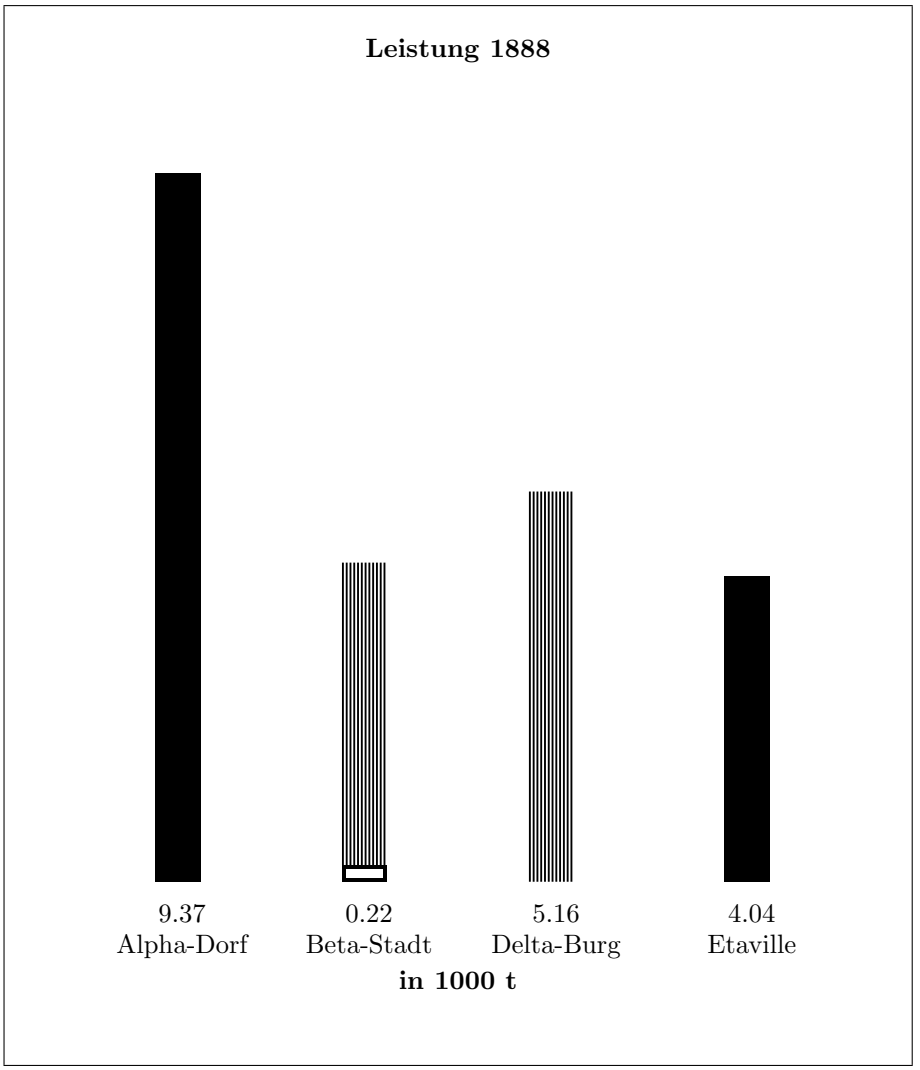
```
%
%  3.b.  Makro für einen hohlen Balken,
%        der oben auf einen anderen gesetzt wird
%
\def\hbalkentop##1{\kern-\dimen1{%
  \dimen0=##1#1\advance\dimen0 by \dimen3
  \vrule height\dimen0 width0.05cm depth-\dimen3
  \dimen4=0.05cm\advance\dimen4 by \dimen3
  \kern-0.05cm
  \vrule height\dimen4 width\dimen1 depth-\dimen3
  \dimen0=##1#1\advance\dimen0 by \dimen3
  \kern-0.05cm
  \vrule height\dimen0 width0.05cm depth-\dimen3
  \kern-\dimen1
  \dimen0=##1#1\advance\dimen0 by \dimen3
  \dimen2=\dimen0\advance\dimen2 by-0.05cm
  \vrule height\dimen0 width\dimen1 depth-\dimen2}%
  \advance\dimen3 by ##1#1}%
%
%   4.a  Makro für einen schraffierten Balken
%
\def\sbalken##1##2{\markier{##2}%
   \dimen3=##1#1%
   \hbox to \dimen1{%
     \leaders
     \hbox{\vrule width0.025cm height##1#1\hskip0.025cm}\hfill}}%
%
%   4.b  Schraffierter Balken oben aufgesetzt
%
\def\sbalkentop##1{\kern-\dimen1{%
  \dimen0=##1#1\advance\dimen0 by \dimen3
  \hbox to\dimen1{%
   \leaders\hbox{\vrule width0.025cm height\dimen0 depth-\dimen3
     \hskip0.025cm}\hfill}}%
  \advance\dimen3 by ##1#1}%
%
%  Zusammensetzen des Diagramms
%
\vbox to #3#1{%
\bigskip
\ifx#4\void\else\hbox to #2#1{\hss#4\hss}\fi
\vfill
\hbox to #2#1{#6\hss}\vskip1cm
\ifx#5\void\else\hbox to #2#1{\hss#5\hss}\vskip1cm\fi
\vss}}}
```

Die Aufrufe für Diagramme haben dann die folgende Gestalt:

```
\def\hilf#1#2{\hbox{\setbox0=\hbox{#2}
              \vtop{\hbox to \wd0{\hfil#1\hfil}\hbox{#2}}}}
\newdimen\mydimen \mydimen=0.3cm
\centerline{\Einrahmung{%
   \diagramm{\mydimen}
            {35}
            {18}
            {\bf Leistung 1888}
            {\bf in 1000 t}
            {\neubreite{0.6}
             \hfill \bbalken {9.37} {\hilf{9.37}{Alpha-Dorf}}
             \hfill \hbalken {0.22} {\hilf{0.22}{Beta-Stadt}}
             \hfill \sbalken {5.16} {\hilf{5.16}{Delta-Burg}}
             \hfill \bbalken {4.04} {\hilf{4.04}{Etaville}}
             \hfill}}}
\centerline{\Einrahmung{%
   \diagramm{cm}
            {12}
            {14}
            {\bf Leistung 1888}
            {\bf in 1000 t}
            {\neubreite{0.6}
             \hfill \bbalken {9.37} {\hilf{9.37}{Alpha-Dorf}}
             \hfill \hbalken {0.22} {\hilf{0.22}{Beta-Stadt}}%
                    \sbalkentop{4}
             \hfill \sbalken {5.16} {\hilf{5.16}{Delta-Burg}}
             \hfill \bbalken {4.04} {\hilf{4.04}{Etaville}}
             \hfill}}}
```

und liefern dann





Der erste Aufruf von "`\diagramm`" arbeitet dabei mit einer eigenen Skalierungsgröße "`\mydimen`", die kurz vorher definiert wird. Dort erfolgen die Längenangaben in Einheiten von 0.3 cm.

12.2 Deutsche Anführungszeichen

Bemerkung: In folgendem Beispiel werden die Anführungszeichen zusammengesetzt. Dies ist notwendig, wenn keine neuen 256-er Zeichensätze verwendet werden. Ebenso werden dort durch die Befehle `<<` und `>>` die *"guillemtes"* direkt erzeugt.
Im deutschen Textsatz wird die gesprochene Rede in Anführungszeichen gesetzt. Ob dies nun notwendig oder auch sehr schön ist, ist eine andere Frage. Da sind zuerst die deutschen ‹Gänsefüßchen›, am Anfang zwei Kommas « „ » und am Ende zwei umgedrehte Kommas « “ », die oben aufgehängt sind. Daneben gibt es noch die « französischen Gänsefüßchen » (*guillemets).* In Deutschland zeigen sie meist mit den

Spitzen nach »innen«, in der Schweiz müssen sie mit den Spitzen nach «außen» zeigen.

Bisher wurden diese Zeichen beim Satz dieser Schrift nicht so verwendet. Es sind nur amerikanische Akzente und Anführungszeichen oben verwendet worden. Mit Hilfe der folgenden Makros lassen sich deutsche Anführungszeichen leicht verwenden:

```
\let\less=<
\let\greater=>
\def\komma{,}
\def\textless{\leavevmode
              \raise1pt\hbox{$\scriptscriptstyle<$}}
\def\textlless{\leavevmode
              \raise1pt\hbox{$\scriptscriptstyle\ll$}}
\def\textgreater{\leavevmode
              \raise1pt\hbox{$\scriptscriptstyle>$}}
\def\textggreater{\leavevmode
              \raise1pt\hbox{$\scriptscriptstyle\gg$}}
%%
\catcode`\,=\active % ACHTUNG! \ifdim ... < ... usw.
\catcode`\<=\active %          gehen anschließend
\catcode`\>=\active %          nicht mehr. Makros mit
\catcode`\?=\active %          <> - Abfragen sind
\catcode`\!=\active %          davor zu definieren.
%%
\def\ignore#1{}
\def?{\char"3F{\kern0pt}}
\def!{\char"21{\kern0pt}}
\def,{\komma\futurelet\next\commatest}
\def\commatest{\ifmmode\else
                  \ifx\next,\kern-.11em\fi
               \fi}
\def<{\futurelet\next\lesstest}
\def>{\futurelet\next\greatertest}
\def\lesstest{\ifmmode \less  \let\next=\relax
              \else
                 \ifx\next<\textlless \let\next=\ignore
                 \else \textless  \let\next=\relax\fi
              \fi\next}
\def\greatertest{\ifmmode \greater  \let\next=\relax
                 \else
                 \ifx\next>\textggreater \let\next=\ignore
                  \else \textgreater \let\next=\relax\fi
              \fi\next}
```

Die Idee dahinter ist, daß beim Auftreten eines Kommas oder kleiner-größer-Zeichens geprüft wird, ob dahinter wieder das gleiche Zeichen folgt. Wenn dies der Fall ist, werden die beiden Zeichen ein wenig zusammengerückt oder für beide ein anderes Symbol gesetzt.

Die merkwürdige Umdefinition von "?" und "!" hat folgenden Grund: Die Zeichenfolgen ?‘ und !‘ sind als Ligaturen eingetragen, sie liefern im Normalfall die spanischen Satzzeichen "¿" und "¡". Der Befehl `\catcode‘\?=\active` deklariert das Fragezeichen als Befehlssymbol. Damit kann dieses in einem Makro eine neue Wirkung zugeteilt bekommen. Das Makro `\def?{\char"3F{}}` gibt dann nur das Zeichen aus der Codetabelle aus; die Befehle "`{\kern0pt}`" bewirken, daß das folgende Zeichen nicht mehr zu der Ligatur zusammengezogen werden kann. Für die Anführungszeichen *unten* werden also Kommata und für *oben* Akzente — ‘ — eingegeben. Der zusätzliche Befehl "`\leavevmode`" sorgt dafür, daß der folgende Text im *horizontal mode* gesetzt wird.

Das Beispiel

```
<< Rotk\"appchen, wohin gehst du? >> sprach der Wolf.
>> Rotk\"appchen, wohin gehst du? << sprach der Wolf.
,,Rotk\"appchen, komm mit mir!‘‘ sprach der Wolf.
 ,Rotk\"appchen, komm mit mir!‘  sprach der Wolf.
```

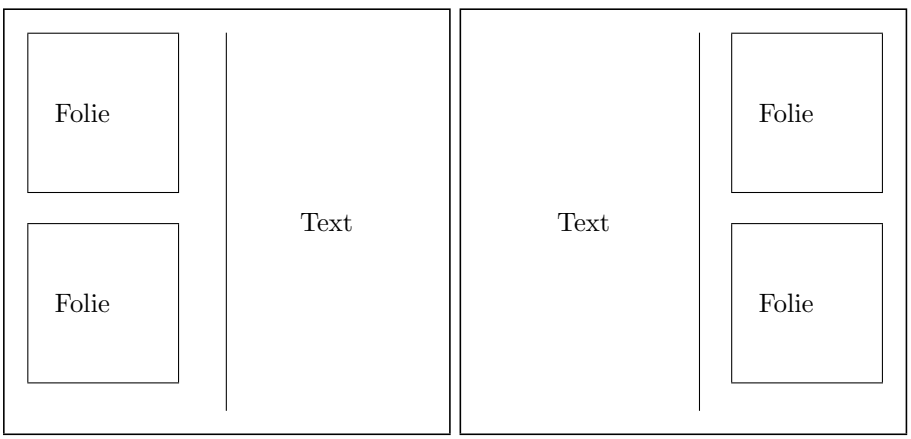
liefert dann als Ausgabe:

≪Rotkäppchen, wohin gehst du?≫ sprach der Wolf.
≫Rotkäppchen, wohin gehst du?≪ sprach der Wolf.
„Rotkäppchen, komm mit mir!“ sprach der Wolf.
‚Rotkäppchen, komm mit mir!‘ sprach der Wolf.

Übrigens bei genauem Hinsehen wird man feststellen, daß die Befehle "<" und ">" im Mathematikmodus ihre alte Bedeutung behalten haben.

12.3 Vorlesungsskript

Im folgenden Beispiel soll ein Layout für ein Skriptum vorgestellt werden, in dem die in einer Vorlesung vorgestellten Folien mit zusätzlichem Kommentar dargestellt werden. Die dazu notwendigen Makros werden so gestaltet, daß einmal das Skript und das andere Mal die Vorlagen für Folien ausgegeben werden können. Die Ausgabe soll dann etwa die folgende Form besitzen:



1 Motivation
2 Projektorganisation
3 Programmentwurf
  3.1 Modulares Programmieren
  3.2 *bottom up / top down*
  3.3 Datenstrukturen
4 Programmerstellung
  4.1 *programming tools*
  4.2 Strukturierte Programmierung
  4.3 Portabilität
  4.4 Programmierstil
  4.5 Defensive Programmierung
5 Testen
6 *debugging* — 'Entlausung'

Fragebogen

1 Welche Eigenschaften hat Ihrer Meinung nach ein *gutes* Programm?

2 Welche Eigenschaften hat Ihrer Meinung nach ein *guter* Programmierer?

In dieser Vorlesung sollen Kenntnisse vermittelt werden, die *gutes Programmieren* auszeichnen.

Antworten zu *gutem Programm*:

17 × ist lesbar (für Menschen)
12 × ist übersichtlich aufgebaut
12 × läuft
9 × erfüllt die gestellte Aufgabe
8 × berücksichtigt mögliche

Bedienfehler
8 × ist leicht korrigierbar
8 × läuft schnell
7 × ist gut dokumentiert
6 × ist ausbaufähig
5 × ist der Maschine angepaßt
5 × braucht wenig Speicher
4 × enthält Kommentare
2 × ist leicht zu bedienen
1 × ist portabel
1 × ist testbar

Die dazu nötigen Makros lauten:

```
\newdimen\papersize          % Gesamtpapierbreite
\newdimen\columnsize         % Breite der Textspalte
\newdimen\folienhsize        % Breite der Folie
\newdimen\folienvsize        % Höhe der Folie
\newbox\upperbox             % Box für die obere Slide
\newbox\lowerbox             % Box für die untere Slide
%
\newif\ifslides              % \slidestrue -> \ifslides := true
                             % \slidesfalse -> \ifslides := false
                             % Nach >> \slidestrue << werden nur noch die
                             % Slides ausgegeben.
%
\papersize=\hsize
\columnsize=0.5\hsize
%
\folienhsize=\hsize
\advance\folienhsize by - \columnsize
\advance\folienhsize by - 3em           % \qquad
%
\folienvsize=0.4\vsize
%
\hsize=\columnsize
%
\def\makeheadline{\vbox to 0pt
                  {\vskip -22.5pt
                   \hsize=\papersize
                   \line{\vbox to 8.5pt{}\the\headline}%
                   \vss}%
                  \nointerlineskip}
%
\def\makefootline{\baselineskip=24pt\hbox to \papersize{\the\footline}}
\def\gesamtoutput{\shipout\vbox{\makeheadline
                                \makepagebox
                                \bigskip
                                \bigskip
                                \makefootline}%
                  \advancepageno
                  \ifnum\outputpenalty>-20000
                      \else\dosupereject\fi}
```

```
%
% Makro um etwas einzurahmen
%
%  #1 <- Abstand des Rahmens
%  #2 <- Dicke des Striches
%  #3 <- Information
%
\def\rahmen#1#2#3{\vbox{\hrule height #2
                        \hbox{\vrule width #2
                              \hskip#1
                               \vbox{\vskip#1{}#3\vskip#1}%
                               \hskip#1
                               \vrule width#2}%
                        \hrule height#2}}

%
% Liefert eine umrahmte Folie zurück
%
\def\rahmenbox#1{\vbox to 0.5\vsize
                  {\ifvoid#1\vfil\else
                     \rahmen{0.2cm}{0.4pt}{\box#1}\vfil\fi}}
\def\makepagebox{\hbox to \papersize
                 {\ifodd\pageno
                    \vtop to \vsize{\hrule height 0pt
                                    \boxmaxdepth=\maxdepth
                                    \pagecontents\vfil}%
                    \hfil
                    \vtop to \vsize{\hrule height 0pt
                                    \rahmenbox\upperbox
                                    \nointerlineskip
                                    \rahmenbox\lowerbox}%
                   \else
                     \vtop to \vsize{\hrule height 0pt
                                     \rahmenbox\upperbox
                                     \nointerlineskip
                                     \rahmenbox\lowerbox}%
                     \hfil
                     \vtop to \vsize{\hrule height 0pt
                                     \boxmaxdepth=\maxdepth
                                      \pagecontents}%
                     \fi}}
```

```
\def\slideoutput{\ifvoid\upperbox\else
                    {\count1=1\shipout\vbox{\rahmenbox\upperbox}}\fi
                 \ifvoid\lowerbox\else
                    {\count1=2\shipout\vbox{\rahmenbox\lowerbox}}\fi
                 \advancepageno
                 \setbox0=\vbox{\box255}}
%
\output={\ifslides\slideoutput\else\gesamtoutput\fi}

%
%   \endslide wird nach \beginslide neu definiert.
%   Auf diese Weise wird ein Fehlermeldung erzeugt,
%   falls \endslide ohne vorheriges \beginslide
%   aufgerufen wird.
%

\def\endslide{\errmessage{beginslide fehlt}}

\outer\def\beginslide{
  \ifvoid\upperbox
    \begingroup
    \def\endslide{\vfil\egroup\endgroup}
    \global\setbox\upperbox
        \vbox to \folienvsize\bgroup\hsize=\folienhsize
  \else
    \ifvoid\lowerbox
     \begingroup
     \def\endslide{\vfil\egroup\endgroup}
     \global\setbox\lowerbox
         \vbox to \folienvsize\bgroup\hsize=\folienhsize
    \else
      \vfill\eject
      \begingroup
      \def\endslide{\vfil\egroup\endgroup}
      \global\setbox\upperbox
         \vbox to \folienvsize\bgroup\hsize=\folienhsize
   \fi
 \fi}
```

Dazu noch einige Erläuterungen:

1. Die Steuervariable `\ifslides` regelt, ob nur Folien oder der gesamte Text ausgegeben werden soll. Nach einem Befehl `\slidestrue` werden *nur* die Folien in einzelne Ausgabeseiten geschrieben. Die Numerierung erfolgt mit Hilfe des Zählregisters `\count1`, das jeweils noch die erste und zweite Folie einer Seite durchnumeriert. Die Seitenzählung lautet dann "1.1, 1.2, 2.1, 2.2, ...", wobei dann '2.2' die zweite Folie auf Seite 2 bezeichnet.
2. Die Variable `\papersize` enthält die ursprüngliche Breite der Seite (`\hsize`); `\columnsize` ist die Breite der Textspalte, `\folienhsize` und `\folienvsize` sind Breite und Höhe einer einzelnen Folienbox.

3. Die Information für eine einzelne Folie wird zwischen den beiden klammernden Befehlen `\beginslide` und `\endslide` angegeben.
   Zunächst wird die obere Box einer Seite besetzt, dann die untere. Sind allerdings beide Boxen belegt, erfolgt zwangsweise ein Seitenvorschub.

Die Makros sind unter Umgestaltung der Output-Routinen konstruiert. Die Funktionen `\headline` und `\footline` arbeiten dabei in der gewohnten Weise und setzen jeweils ihren Inhalt über die *gesamte* Seitenbreite. Fußnoten werden allerdings nur bezüglich der Textspalte verwaltet. In dieser wird der normale Fließtext wie gewöhnlich verarbeitet, jedoch mit geringerer Zeilenlänge gesetzt.

Die Eingabe für das vorangehende Beispiel hat die Form:

```
\beginslide
\raggedright
\item{1}Motivation
\item{2}Projektorganisation
  . . .
\item{6} {\it debugging} --- ‘Entlausung’
\endslide
%
%
\beginslide
\centerline{\bf Fragebogen}
\raggedright
\medskip
\item{1} Welche Eigenschaften hat Ihrer Meinung nach ein
   {\it gutes} Programm?
  . . .
\endslide
%
%
\noindent
In dieser Vorlesung sollen Kenntnisse vermittelt werden, die
  . . .
\leftline{Antworten zu {\it gutem Programm}:}
\medskip
{\obeylines
\itemitem{17 $\times$}   ist lesbar (f\"ur Menschen)
\itemitem{12 $\times$}   ist \"ubersichtlich aufgebaut
 . . .
\vfill\eject
```

13 Datenorganisation

13.1 Standarddateien

Während eines Programmlaufes werden durch das TEX-Programm eine Reihe von Dateien bearbeitet, deren Inhalt und deren Namensgebung einigen Konventionen unterliegt.

TEX stattet, soweit es das jeweilige Betriebssystem zuläßt, Dateien mit bestimmten Funktionen mit einer speziellen Namenserweiterung (*extension*) aus. Diese besteht aus drei Buchstaben, welche den jeweiligen Inhalt kennzeichnen.

So wird etwa ein "`.TEX`" (MS-DOS,UNIX,...) oder auch ein "`_TEX`" (NOS/VE) am Ende des Dateinamens für Eingabedateien erwartet. Die Konventionen für diese "Anhängsel" sind wie folgt:

`.TEX` für Eingabedateien — etwa bei "`\input`" — oder auch bei Ausgabedateien (mittels "`\write`")
Fehlt die Angabe der *extension* beim Aufruf, so wird diese automatisch angefügt. Die Befehle

```
\input DATEN
\input DATEN.TEX
```

geben damit die gleiche Datei an.

`.FMT` für eigene *format*-Dateien
Die Formatdatei wird beim Programmstart ausgewählt. Typische Anwahlen sind "`&plain`" oder "`&latex`" (siehe auch Abschnitt 2.5).
Die Erstellung der *format*-Dateien geschieht mit dem Programm "`INITEX`".

`.LOG` für die Protokolldatei *(log-file)*

`.DVI` für die Ausgabedatei mit der eigentlichen Satzinformation *(device independent file)*, die durch den jeweiligen Gerätetreiber ausgewertet wird.

`.TFM` für eine Font-Metrik-Datei, die die Information über die verschiedenen Schriften enthält. Diese wird beim "`\font`"-Befehl angesprochen.
So enthält etwa "`cmssi10.tfm`" die metrische Information für die Schrift 'cmssi10' (sans serife italic 10 pt), deren Anwahl etwa durch den Befehl "`\font\sani=cmssi10`" erfolgt.

Erfolgt die Eingabe beim Programmstart interaktiv, so wird als Standardbesetzung für den Auftragsnamen (`\jobname`) ein "`TEXPUT`" gesetzt. Wird dagegen in der Anfangsanfrage der Name einer Eingabedatei angegeben, aus der die Satzinformation gelesen wird, erfolgt die Benennung der Ausgabedateien entsprechend diesem Namen. Lediglich werden am Ende des Namens andere Kürzel angehängt. Heißt die Eingabedatei beispielsweise "`TEXDATEN.TEX`", so erhält die Protokolldatei den Namen "`TEXDATEN.LOG`" und die Ausgabedatei wird "`TEXDATEN.DVI`" genannt. Wie gesagt, das Trennzeichen "." braucht nicht in jeder Implementierung so verwendet zu werden, es kann auch ein anderes Zeichen sein.

13.2 Organisation der Eingabe

Bei der Bearbeitung eines größeren Projektes oder bei regelmäßig wiederkehrenden gleichartigen Arbeiten empfiehlt sich, die Eingabedaten strukturiert nach einem Organisationsschema zu verwalten. Die natürliche Form ist die Aufteilung der Information in verschiedene Eingabedateien. Zusätzlich können diese durch die Ablage in verschiedenen Verzeichnissen weiter gegliedert werden. Nahezu alle TEX-Implementierungen sind in der Lage, eine bestimmte Eingabedatei in mehreren Verzeichnissen zu suchen.

Durch den "`\input`"-Befehl wird dann die jeweils einzulesende Datei angegeben.

Makro-Gliederung

Die wesentliche Gliederung ist zunächst die Aufteilung der später zu benutzenden eigenen Makrobefehle. Hier sind die Unterteilung in einen Grundschatz, sozusagen die private Erweiterung oder Abänderung von plain-TEX-Makros, und die verschiedenen privaten Zusatzpäckchen sehr zu empfehlen. Die Zusatzmakros werden dann je nach Aufgabe mit einem `\input` Befehl eingelesen und aktiviert.

Bei der Erstellung eigener Befehlsgruppen sollte man darauf achten, daß es möglichst zu keinen Überschneidungen zwischen den verschiedenen Makropaketen durch doppelte Namensvergaben kommt.

Beispiele solcher Gruppen sind etwa Befehle für die Mathematikeingabe, die eigenen Wünschen angepaßt wird. Diese können etwa abgekürzte Befehle für griechische Buchstaben, vorbereitete Anweisungen für bestimmte Matrizen und Integralformen enthalten. Eine Makrogruppe kann aber auch die Definition des Layouts einer hauseigenen Telefonliste oder die Festlegung des eigenen Fußnotenformats enthalten.

Als praktisch hat sich herausgestellt, Befehle, die in irgendeiner Form das äußere Layout beeinflussen, etwa die Größe des Satzspiegels, den Standardzeilenabstand, möglichst nur an einer Stelle zusammengefaßt zu halten. Dann sind alle Daten für spätere Formatänderungen leichter zu korrigieren.

Gliederung der Eingabedaten

Das zweite Organisationsschema betrifft die *Eingabedaten*. Diese Aufteilung hängt natürlich sehr von der betreffenden Aufgabenstellung ab. Aus der Praxis lassen sich jedoch einige Ratschläge erteilen: Bei größeren Informationsmengen, etwa für ein Buch, sollte man in jedem Fall eine Unterteilung in einzelne Kapitel vollziehen. Längere Kapitel werden sogar in mehrere Dateien aufgeteilt. Dadurch können die einzelnen Abschnitte separat bearbeitet werden. Am Ende werden sie durch eine alles aufrufende

Eingabedatei zusammengefaßt, in der dann beispielsweise die Befehle

```
\input makros
\input inhalt
\input kapitel1
\input kapitel2
\input kapitel3
...
```

enthalten sind. Häufig wird man auch, vom satztechnischen Standpunkt aus, komplexe Eingaben getrennt halten. Kandidaten hierfür sind komplexe Tabellen, womöglich mit komplizierterer Mathematik gemischt. Da es möglich ist, `\input`-Befehl mehrstufig zu verwenden — die Datei `KAPITEL1.TEX` mag etwa folgende Anweisungen enthalten

```
% Kapitel 1
\input kap1a
\input kap1b
\input kap1c
```

— kann dies sehr bequem geschehen.

13.3 INITEX

In vielen Implementierungen ist eine spezielle Variante des TEX-Programms vorhanden: INITEX. Diese bietet die Möglichkeit, *format*-Dateien zu erstelllen. Das Programm erhält als Eingabe *alle* Makros, die später bekannt sein sollen, und die sprachspezifischen Trennmuster, nach denen dann getrennt werden wird.

Selbstverständlich können die so vorbereiteten Befehle im TEX-Lauf wieder überdefiniert und mit neuen Bedeutungen versehen und durch weitere Anweisungen ergänzt werden — wie es hier schon die ganze Zeit mit den Makros von *plain*-TEX praktiziert wird.

Die Ausgabe von INITEX — mittels des "`\dump`" Befehls — besteht in der bereits erwähnten *format*-Datei. Der Vorteil dieses Verfahrens liegt in der Zeitersparnis bei der späteren Verwendung. Wird nämlich beim Programmstart des normalen TEX-Programms — siehe Abschnitt 2.5 — der Name dieser Formatdatei angegeben, so werden die bereits vorinterpretierten Makros sehr schnell eingelesen, ohne daß die üblichen komplexen Abprüfungen stattfinden. Das gleiche gilt für alle Schriftdefinitionen durch "`\font`"; diese sind bereits fertig vollzogen. Die zugehörigen *tfm*-Dateien werden später nicht noch einmal eingelesen.

So liegen die Standard-Makros des plain-TEX normalerweise in einer Datei mit dem Namen "`PLAIN.TEX`" und die Trennungen für die englische Sprache* in "`HYPHEN.TEX`". Trennmuster werden durch den Befehl "`\patterns{...}`" eingegeben. Das Laden der Trenntabelle sollte nicht vergessen werden, sonst trennt das Programm nämlich gar nicht!

Am Aufbau von "`PLAIN.TEX`" kann man sich dann für eigene Anwendungen auch gut orientieren. So werden durch einen "`\input hyphen`" Befehl die Trennungen innerhalb von "`PLAIN.TEX`" eingelesen.

---

* Deutsche Trennmuster wird man u. U. in `GHYPHEN.TEX` finden.

Die Erstellung der eigenen *format*-Datei geschieht, nachdem alle Makros definiert sind, durch den `\dump`-Befehl. Im Protokoll wird der Name der dann erstellten Datei gemeldet. Diese besitzt dann die Namenserweiterung "`.FMT`".

Einen kleinen Haken hat die Arbeit mit eigenen *format*-Dateien allerdings: Diese Dateien sind *nicht* zwischen verschiedenen Implementierungen austauschbar!

Auch soll bedacht werden, daß natürlich durch die geladenen Makros Speicherplatz belegt wird. Dieser kann dann später bei der Anwendung fehlen, weil sehr viel Raum durch eigentlich nicht benötigte Makros besetzt ist.

13.4 Zugriff auf weitere Klartextdateien

Die normale Ausgabe eines TEX-Laufes ist die *dvi*-Datei mit der Satzinformation zur Wiedergabe auf einem Drucker oder Bildschirm. Daneben kann mit Hilfe der folgenden Befehle auch *Klartextinformation* in Hilfsdateien geschrieben oder aus ihnen gelesen werden. Eine knappe Übersicht bietet die folgende Befehlstabelle:

Eingabe	*Ausgabe*	*Wirkung*
`\newread \`*name*	`\newwrite \`*name*	freie Dateinummer zuteilen
`\openin` *n* `=` *dateiname*	`\openout` *n* `=` *dateiname*	Datei öffnen
`\read` *n* `to \`*ziel*	`\write` *n* `{ . . . }`	lesen / schreiben
`\closein` *n*	`\closeout`*n*	Datei schließen
`\ifeof` *n*		Dateiende (Eingabe) prüfen

Einer Datei muß vor dem ersten Lese- oder Schreibzugriff eine Stromnummer zwischen 0 und 15 zugeordet werden. Diese wird bei den Befehlen `\openin` und `\openout` als "*n*" dem Dateinamen zugeordnet. Bei allen weiteren Ein- oder Ausgabebefehlen `\read`, `\write` usw. muß sie mitangegeben werden. Wird eine solche Stromnummer verwendet, ohne daß vorher mittels `\openin` oder `\openout` eine Dateiöffnung erfolgte, so wird die Ausgabe in den *log*-File umgelenkt, sowie auf dem Terminal ausgegeben. Die Ausgabe auf dem Terminal unterbleibt bei einer negativen Dateinummer.

```
\write -1 {An Testposition vorbei}
\write 66 {Vor der kritischen Stelle}
```

gibt den Text "`Vor der kritischen Stelle`" sowohl auf dem Bildschirm als auch in den *log*-File aus. Dagegen wird "`An der Testposition vorbei`" nur in den *log*-File geschrieben. Man kann nun ohne weiteres die Zuordnung Dateinummer zu Dateiname direkt setzen, wie etwa in "`\openin 7 = DATEN`". Besser ist jedoch eine Verwaltung der freien Dateinummern durch TEX. Mit den Befehlen `\newread` und `\newwrite` wird eine freie Nummer auf einem symbolischen Namen zugeteilt. Anschließend wird diesem symbolischen Namen in `\openin` oder `\openout` der Dateiname zugeordnet. Also statt

```
\openin 7 = DATEN
\read 7 to \MeineDatenZeile
\openout 12 = AUSGABE
\write 12 {Ich bin auf Seite \folio}
```

sollte man besser

```
\newread\DatenFile   \newwrite\AusgabeFile
\openin\DatenFile=DATEN
\read\DatenFile to \MeineDatenZeile
\openout\AusgabeFile= AUSGABE
\write\AusgabeFile{Ich bin auf Seite \folio}
```

verwenden. Daß die Verwendung eines symbolischen Namens von einem besseren Arbeitsstil zeugt, ist eindeutig klar. Zusätzlich werden durch seine Benutzung spätere Änderungen, wie die Kombination mit weiteren Makros, erleichtert.

13.5 Eingaben aus Klartextdateien

Durch den Befehl `\newread\DatenFile` sei ein symbolischer Name einem bestimmten Eingabekanal zugeordnet. Anschließend ist die zu lesende Datei zu eröffnen. Bei der Eröffnung mit `\openin` wird dem TEX-Programm der Name der Eingabedatei mitgeteilt. Durch

```
\openin\DatenFile=DATEN
```

wird die Zuordnung auf die externe Datei "`DATEN.TEX`" oder evtl. auch "`DATEN_TEX`" getroffen. Welche Namenserweiterung "`.TEX`" oder ähnliches nun angefügt wird, ist von der jeweiligen Implementierung abhängig. Es gelten hier die gleichen Regeln wie beim Befehl `\input`.

Wird die Anweisung `\openin` jedoch nicht gegeben, so werden die späteren Eingaben am Terminal verlangt.

Die tatsächliche Eingabe erfolgt durch einen Befehl wie

```
\read\DatenFile to \DatenZeile
```

der zunächst *eine Zeile* aus der Datei "`DATEN.TEX`" auf `\DatenZeile` einliest. Dies hat die gleiche Bedeutung wie die Definition eines Makros `\DatenZeile` ohne Parameter. Angenommen, in der Datei `DATEN.TEX` steht die folgende Information:

```
Sch\"afer, Anton
Meier, Willi
Schwarz, Norbert
```

Dann haben die drei "`\read\DatenFile to \DatenZeile`" Aufrufe die gleiche Wirkung wie die folgenden drei Makrodefinitionen:

```
\def\DatenZeile{Sch\"afer, Anton }
\def\DatenZeile{Meier, Willi }
\def\DatenZeile{Schwarz, Norbert }
```

Damit ist nun auch leicht erklärbar, daß beim Einlesen mit einem `\read`-Befehl Klammerstrukturen beachtet werden. Geht ein Klammergebirge in einer Eingabezeile nicht zu, so werden weitere Eingabezeilen eingelesen, bis die Klammern aufgehen. Man beachte, daß für das Zeilenende ein zusätzliches Leerzeichen am Makroende abgespeichert wird. Welches Zeichen für das Zeilenende übernommen wird, regelt das interne Register `\endlinechar`. Dieses enthält die ASCII-Codenummer des einzusetzenden Zeichens. Ist der Wert kleiner als Null, so wird kein Zeichen eingefügt.

Durch den Befehl `\ifeof\DatenFile` kann in einem Makro sehr leicht getestet werden, ob das Dateiende bereits erreicht ist.

Wird nun ein `\read`-Befehl auf eine nicht eröffnete Datei gegeben, so wird vom Terminal gelesen. Falls die angegebene Dateinummer ≥ 0 war, wird die Bezeichnung des Zielmakros, in unserem Beispiel `\DatenZeile`, als Anfrage

```
\DatenZeile=
```

gestellt. Bei einer negativen Dateinummer etwa in "`\read -1 to \DatenZeile`" wird zwar in gleicher Weise vom Terminal gelesen, der Anfragetext wird jedoch unterdrückt.

Soll die gleiche Strom-, bzw. Dateinummer für verschiedene Dateien nacheinander verwendet werden, muß beispielsweise durch `\closein\DatenFile` zuvor die Eingabedatei geschlossen werden. Ein neuer `\openin`-Befehl mit `\DatenFile` als Stromnummer eröffnet dann eine Datei neu zum Lesen.

13.6 Ausgaben in Klartextdateien

Das Lesen zusätzlicher Dateien mit `\read` usw. war noch recht einfach gegenüber den Punkten, die bei Schreiboperationen zu beachten sind. Das Wichtigste soll vorneweg notiert sein:

> **Ein Schreibbefehl `\openout`, `\write` oder `\closeout` wird nicht sofort ausgeführt, wenn er als Befehl gelesen wird. Er wird erst dann tatsächlich ausgewertet, wenn die Druckseite, auf der er gefunden wurde, durch die Output-Routine auch in den dvi-File geschrieben wird.**

Dieses zunächst sinnlos erscheinende Verhalten hat seinen guten Grund: Erst während der Output-Routine stehen die typischen Daten, wie die Seitennumerierung, die man sich gerne für ein Stichwortverzeichnis oder ein Inhaltsverzeichnis merken möchte, fest.

Auf der anderen Seite kann durch den Befehl `\immediate`, welcher den Befehlen `\openout`, `\write` oder `\closeout` vorangestellt wird, auch eine sofortige Ausführung des folgenden Schreibbefehls erzwungen werden.

Die verzögerte Ausführung eines `\write`-Befehls hat nun aber auch die Auswirkung, daß eventuelle Makros, die als Information in diesem Befehl stehen, erst während der Output-Routine expandiert werden. Dann kann sich aber die Bedeutung etwa einer gespeicherten Absatzüberschrift schon wieder verändert haben.

Die beste Verhaltensweise hierbei ist, zunächst bei der Definition von Makros mittels `\immediate` dafür Sorge zu tragen, daß die Ausgabedatei sofort eröffnet wird. Also etwa in der Befehlsfolge

```
\newwrite\IndexDatei
\newwrite\InhaltDatei
\immediate\openout\IndexDatei=INDEX.TEX
\immediate\openout\InhaltDatei=INHALT.TEX
```

zwei Dateien für ein Stichwortregister und ein Inhaltsverzeichnis eröffnen. Ein Stichwortregister ist dann leicht durch `\write`-Befehle zu realisieren:

```
\write\IndexDatei{Fehler \folio}
\write\IndexDatei{Konvergenz \folio}
```

Man wird dann in der Ausgabedatei `INDEX.TEX` etwa die folgende Information wiederfinden:

```
Fehler 17
Konvergenz 18
```

Für eine spätere Weiterverarbeitung ist es ratsam, gleich Makroaufrufe mit in die Ausgabedatei zu schreiben. Da die Ausgabe satzweise erfolgt, treten bei einer späteren

Sortierung auch keine Probleme auf. Der Befehl `\string` gibt den Namen des folgenden Befehls im Klartext aus. Durch `\string\Objekt` wird die Zeichenfolge "`\Objekt`" in die Ausgabedatei geschrieben.

```
\write\IndexDatei{\string\Objekt:Fehler \string\Seite \folio}
\write\IndexDatei{\string\Objekt:Konvergenz \string\Seite \folio}
```

Natürlich wird ein solcher Befehl für einen Stichworteintrag in einem Makro

```
\def\Index#1{\write\IndexDatei{\string\Objekt:#1
                               \string\Seite \folio}}
```

mit den Aufrufen `\Index{Fehler}` und `\Index{Konvergenz}` versteckt. In der Ausgabedatei erhält man dann beispielsweise die Information:

```
\Objekt:Fehler \Seite17
\Objekt:Konvergenz \Seite18
```

Nach einer Sortierung kann mittels `\input` diese Datei dann direkt eingelesen und verarbeitet werden, vorausgesetzt natürlich, es existieren dazu passende Makros `\Objekt` und `\Seite`. Da bei diesen Ausgaben kein `\immediate` gesetzt wurde, erfolgt die Auswertung des `\write`-Befehls erst innerhalb der Output-Routine.

Schwieriger wird es dagegen, wenn nur ein *Teil* der Information sofort und der andere erst während der Output-Routine expandiert werden soll. Dies ist der Fall, wenn sowohl eine Formelnumerierung, die automatisch erstellt wird, als auch die Seitenzahl ausgegeben werden soll.

Dargestellt sei die Bewältigung dieses Problems an folgendem Beispiel: Zwei Bestandteile, etwa eine Numerierung und eine Bezeichnung sollen mit ihren aktuellen Werten, die sich auf der Seite auch mehrfach ändern dürfen, ausgegeben werden. Die Seitennummer soll so ausgegeben werden, wie sie in der Output-Routine bei der effektiven Seitenausgabe anfällt. Durch folgende Makros

```
% vorher:  \newcount\FormelNr
%          \newwrite\IndexFile
%          \openout\IndexFile= ???
%          \def\FormelTitel{ ... }
\def\Referenz{%
     \edef\WriteIndex{%
         \write\IndexFile{\string\FormelNr:\number\FormelNr
                          \string\FormelTitel: \FormelTitel
                          \string\Seite: \noexpand\folio}}%
     \WriteIndex}
```

wird mit ungefähr den Aufrufen

```
\advance\FormelNr by 1
\def\FormelTitel{Erg\"anzung}
         . . .
\Referenz
\advance\FormelNr by 1
\def\FormelTitel{Diophantische Gleichung}
         . . .
\Referenz
```

in etwa diese Information erzeugt:

```
\FormelNr:1\FormelTitel: Erg{\accent "7F a}nzung\Seite: 1
\FormelNr:2\FormelTitel: Diophantische Gleichung\Seite: 1
```

Dabei sind noch einige Anmerkungen zu vollziehen: In dem Makro Referenz wird mittels des Befehls `\edef` ein neues Makro `\WriteIndex` definiert, das direkt nach der Definition aufgerufen wird. Durch die Angabe von `\edef`, und nicht `\def`, wird das Makro sofort expandiert. Das heißt, alle weiteren Befehle innerhalb von `\WriteIndex` werden sofort expandiert. Insbesondere wird durch `\number\FormelNr` der aktuelle Wert von `\FormelNr` direkt ausgegeben. Diese direkte Auswertung unterbleibt für die Befehle, denen ein `\noexpand` vorangestellt wird. In diesem Fall ist das nur `\folio`.

Probleme mit Akzenten

Noch ein Hinweis: Wie man an dem Beispiel sieht, werden Makros für Umlaute auch in die TEX-interne Darstellung transformiert. Ein "ä" jetzt allerdings in einer zu sortierenden Datei als "`{\accent"7F a}`" bzw. in der deutschen Fassung noch komplizierter vorzufinden, ist für eine spätere Weiterverarbeitung nicht sehr glücklich. An dieser Stelle kann nun die Information doppelt ausgegeben werden: Einmal in der TEX-Form und einmal in einer sortierfähigen Darstellung. Die folgenden Makros bieten zunächst eine einfache Änderung der Ausgabe, bei der die Umlaute und das Eszett umschrieben ausgeben werden.

```
\def\Umlaut#1{#1e}   % zur Umdefinition von \"
\def\Eszett{ss}      %                      \ss
%
\def\Referenz{%
   {\let\"=\Umlaut\let\ss=\Eszett
    \xdef\SortTitel{\FormatTitel}}
     \edef\WriteIndex{%
         \write\IndexFile{\string\FormelNr:\number\FormelNr
                          \string\FormelTitel: \SortTitel
                          \string\Seite: \noexpand\folio}}%
     \WriteIndex}
```

Der Befehl `\xdef` wirkt wie ein `\global\edef`. Es wird der Inhalt von `\FormatTitel` expandiert. Dabei sind allerdings die Makros `\"` und `\ss` durch die vorangehenden `\let`-Befehle umdefiniert.

Die `\let`-Befehle müssen *vor* dem `\edef`-Makro stehen, da sie *ausgeführt* werden müssen. Die Makros im Definitionsrumpf von `\edef` werden nämlich nur expandiert, aber *nicht ausgeführt*. Insbesondere müssen die Umdefinitionen von `\"` und `\ss` vor der Expandierung erfolgt sein. Zugegeben, die Sache ist an dieser Stelle etwas trickreich. Aber mit der Unterscheidung *Expandierung* und *Ausführung* wieder verstehbar.

Die Aufrufe mit gleicher Information wie oben liefern dann statt der TEX-Form für Umlaute die folgende einfache Ersatzdarstellung.

```
\FormelNr:1\FormelTitel: Ergaenzung\Seite: 1
\FormelNr:2\FormelTitel: Diophantische Gleichung\Seite: 1
```

Die Sortieranforderungen werden häufig unterschiedlich sein. So ist beispielsweise an eine gleiche Behandlung von Umlauten und den entsprechenden Vokalen zu denken. Dies läßt sich aber durch einfache Änderungen in den Hilfsmakros `\Umlaut` oder `\Es-zett` erledigen.

Version 3: Werden Schriften mit 256 Zeichen verwendet, die eine direkte Eingabe der Umlaute und anderer diakritischer Zeichen erlauben, mildern sich diese Probleme drastisch. Eine vernünftige TEX-Implementierung wird dann die Umlaute bei der Ein- und Ausgabe wie ein normales Zeichen behandeln und dem Anwender weitere Umstände ersparen.

Probleme mit anderen Makros

Soll ein Makroname in eine Datei geschrieben werden, so wurde in den bisherigen Beispielen dies durch Voranstellen des Befehls `\string` ermöglicht. Soll jedoch der komplette Inhalt eines Makros, der auch aus weiteren TEX-Befehlen bestehen kann, in eine Datei geschrieben werden, *ohne* daß die Befehle expandiert und ausgeführt werden, ist dies etwas schwieriger und bedarf einiger Vorbereitung. Zunächst wird dargestellt, wie die Definition eines vorhandenen Makros in eine Zeichenfolge verwandelt wird, in alle Zeichen ihre spezielle TEX-Bedeutung verloren haben:

Der Befehl `\meaning` gibt die Bedeutung eines Makros in der folgenden Form aus. Nach beispielsweise `\message{\meaning\bigskip}` erhält man

```
macro:->\vskip \bigskipamount
```

Die beiden Hilfsmakros

```
\def\getdefhelp#1->#2\endhelp{#2}
\def\getdef#1#2{\edef#2{\expandafter\getdefhelp\meaning#1\endhelp}}
```

liefern bei einem Aufruf `\getdef\bigskip\text` auf dem dadurch neu definierten Makro `\text` genau den Text "`\vskip \bigskipamount `", ohne daß der inverse Schrägstrich als Befehlsanfangszeichen interpretiert wird*. Dies sei ausführlich dargestellt. Nach dem Aufruf `\getdef\bigskip\text` stehen folgende Befehle an:

```
\edef\text{\expandafter\getdefhelp\meaning\bigskip\endhelp}
```

Durch `\expandafter` wird die Interpretation von `\getdefhelp` übersrpungen und zuerst `\meaning\bigskip` ausgewertet:

```
\edef\text{\expandafter
  \getdefhelp macro:->\vskip \bigskipamount \endgetdefhelp}
```

Nach der Auswertung von `\getdefhelp`, das die Information zwischen `->` und `\endgetdefhelp` übernimmt, bleibt:

```
\edef\text{\vskip \bigskipamount }
```

* Alle Zeichen, die durch den Befehl `\meaning` generiert werden, sind vom `\catcode` 12 (`\other`). Insbesondere wird "\" nicht mehr als *escape*-Zeichen ausgewertet.

übrig. Dabei werden alle Zeichen im Text “ `\vskip \bigskipamount` ” nur als einfache Textzeichen ausgewertet.

Ist dann beispielsweise ein Befehl

```
\def\Vorspann{\NeuerEintrag Information \Index}
```

gegeben, so wird nach

```
\getdef\Vorspann\TextVorspann
\write\IndexFile{\TextVorspann}
```

die Zeichenfolge“`\NeuerEintrag Information \Index` ” geschrieben.

Ein Hinweis: Durch den verwendeten Befehl `\meaning` wird nach jedem Befehlsnamen ein Leerzeichen erzeugt.

Durch eine geeignete Kombination von Makros, die nur den Befehlstext enthalten, und normalen Befehlen, beispielsweise durch `\write\Indexfile{\TextVorspann \folio \TextNachspann}`, erreicht man die gewünschte gemischte Ausgabe.

14 Anhang

Die im Anhang aufgeführten Referenzen enthalten

1. Eine Kurzbeschreibung aller TEX-Befehle.
   Diese sind mit Querverweisen und zum Teil mit Beispielen versehen. Insbesondere sind die Befehle gekennzeichnet, die für den Mathematiksatz oder den Textsatz gedacht sind.
   Verweise auf Beispiele sind in *kursiven* Seitenangaben verzeichnet.
   Interne TEX-Befehle, die also schon fest einprogrammiert und nicht erst in plain-TEX definiert sind, werden durch ein vorangestelltes " ∗ " markiert.
   Das Zeichen " D " ist jeweils den Originaldefinitionen der Makros vorangestellt.
   Befehle und Makros, die mit der Version 3.0 des TEX-Programms neu hinzugekommen sind, erscheinen mit dem Symbol " 3 ".
2. Eine Aufstellung aller versteckten plain-TEX-Befehle, die im Namen das Zeichen "@" enthalten, und für den Anweder nicht direkt zugreifbar sind.
3. Ein Schlagwortregister.
   Hier werden allerdings die TEX-Befehle nicht noch einmal aufgeführt, sondern nur die Sachverweise.
4. Die Fonttabellen der Computer Modern Schriften.
   Für die wichtigsten Schriften sind die Code-Tabellen der Zeichenbesetzung dargestellt. Dabei werden die schriftspezifischen Font-Parameter mitaufgeführt.
5. Die Fonttabelle zur erweiterten 256-Zeichen-Codebelegung für die westeuropäischen lateinischen Schriften, wie sie auf der TEX-Konferenz in Cork 1990 vereinbart wurde. Diese enthält insbesondere Umlaute und akzentuierte Buchstaben als eigene Zeichen.
6. Ein Literaturverzeichnis.

14.1 Kurzbeschreibung der plain-TEX-Befehle

`\␣`	erzwingt die Ausgabe eines Leerzeichens. `\S \S` liefert "§§", dagegen `\S\ \S` ergibt "§ §"	
`#`	Platzhaltersymbol in Tabellen und Makros	96, 110
`\#`	Textsatz: liefert — # — D `\chardef\#="23`	15
`\$`	Textsatz: liefert — $ — D `\chardef\$="24`	15
`$`	Einleitungssymbol für den Mathematiksatz im Text	66, *73*
`$$`	Einleitungssymbol für den hervorgehobenen Mathematiksatz im *display-style*	70, *73*
`%`	Kommentaranfang: Der Rest der Zeile und der Zeilenwechsel wird ignoriert.	14
`\%`	Textsatz: liefert — % — D `\chardef\%="25`	15, 16
`&`	ist das Tabulatorsymbol zur Trennung der Spalten beim Tabellensatz mit `\halign` oder `\settabs\+ ... \cr`	14, 92, 96
`&&`	Wird das Tabulatorsymbol bei `\halign` in der Musterzeile doppelt gesetzt, so werden die nachfolgenden Musterelemente beliebig häufig, das heißt nach Bedarf, wiederholt.	103, 107
`\&`	Textsatz: liefert — & — D `\chardef\&="26`	15
`'`	Textsatz: (Apostroph) liefert — ' —	16, 69
`\'`	Textsatz: *acute accent* `\'o` liefert — ó — D `\def\'#1{{\accent 19 #1}}`	25
`''`	Textsatz: wirkt wie ein Doppelapostroph — " — (Ligatur)	16
`` ` ``	Textsatz: (Akzent) — ` —	
`` `` ``	Textsatz: (Doppelter Akzent) — " —	16
`` `\ ``	bildet so etwas wie eine *ord*-Funktion. Durch die Eingabe `` `\A `` wird der Wert 65 geliefert. Dies ist z. B. in `\catcode`\!=0` anwendbar, wenn der Code-Wert einer Zahl in Zuweisungen verlangt wird.	
`` \` ``	Textsatz: *grave accent* `` \`o `` liefert — ò — D `\def\`#1{{\accent 18 #1}}`	25
`"`	(Doppelapostroph) liefert — " — Es leitet auch eine hexadezimale Zahl ein, zum Beispiel `\accent "7F`, wobei `"7F"` für 127 steht.	16
`\"`	Textsatz: *Umlaut* `\"o` liefert — ö — D `\def\"#1{{\accent "7F #1}}`	25, *52*

`(` linke runde Klammer — (—, als `\left(` oder `\right(` im Mathematiksatz automatisch wachsend zur Formelgröße. 75

`)` rechte runde Klammer —) —, als `\left)` oder `\right)` im Mathematiksatz automatisch wachsend zur Formelgröße. 75

`[` linke eckige Klammer — [—, als `\left[` oder `\right[` im Mathematiksatz automatisch wachsend zur Formelgröße. 75

`]` rechte eckige Klammer —] —, als `\left]` oder `\right]` im Mathematiksatz automatisch wachsend zur Formelgröße. 75

`{` Gruppenklammer auf: Damit wird ein neuer Block eröffnet oder der Anfang eines Makroparameters gekennzeichnet. Der Befehl `\bgroup` ist zur Eröffnung eines neuen Blocks äquivalent. 14, *68*

`}` Gruppenklammer zu: Damit wird ein Block beendet oder das Ende eines Makroparameters gekennzeichnet. Ein Block kann auch durch den Befehl `\egroup` beendet werden. 14, *68*

`\{` geschweifte linke Klammer im Mathematiksatz "{". Der Befehl hat die gleiche Wirkung wie `\lbrace`. Er kann mit `\left`, `\right` und `\big...` zu wachsenden Klammern kombiniert werden. 15, 75, *77*

D `\def\{{\delimiter "4266308 }`

`\}` geschweifte rechte Klammer im Mathematiksatz "}" (siehe auch `\{`). 15, 75

D `\def\}{\delimiter "5267309 }`

`+` Mathematiksatz: binärer Operator in $1 + 1 = 2$

`\+` beginnt eine Tabulatorzeile. Im Gegensatz zu `\tabalign` kann `\+` nicht innerhalb von Makros verwendet werden. Beispiel: 91ff

```
\settabs 3 \columns
\+eins & zwei & drei \cr
```

(siehe auch `\tabalign`).

D `\outer \def\+{\tabalign }`

`-` Mathematiksatz: binärer Operator in $1 - 1 = 2$ 15
Textsatz: Trennstrich etwa in "O-Beine"

* `\-` Textsatz: Es werden damit alle erlaubten Trennstellen markiert. Beispielsweise wird in `Trenn\-vorschlag` genau eine Trennung gestattet 51
(siehe auch `\hyphenation`).

`--` Textsatz: bis-Strich `12--14 Uhr` ergibt 12–14 Uhr 15

`---` Textsatz: Gedanken—strich 15

`*` Mathematiksatz: binärer Operator $a * b$

`\*` Mathematiksatz: Vortrenner in Produktformeln. 89
`$ (a+b) \* (c+d) \* ... $` liefert, falls die Formel im *text-style* getrennt wird, das Zeichen "×" als Trennzeichen (`\times`)
D `\def\*{\discretionary {\thinspace \the \textfont 2 \char 2}{}{}}`

`/` Schrägstrich — / — 75

* `\/` *italic correction*
liefert im Textsatz eine Abstandskorrektur zum nachfolgenden Text, falls dieser nicht mehr in *italic* ist.
Beispiel: (Man beachte den Abstand bei — *t* t —)
`{\it Font}tabelle` → *Font*tabelle und
`{\it Font\/}tabelle` → *Font*tabelle

`|` Mathematiksatz: normales Zeichen — | — 75
(äquivalent mit `\vert`)
(In Kombination mit `\left`,`\right` und `\big..` ist dieses wachsend; durch `\mid` ist es als Relation verfügbar.)

`\|` Mathematiksatz: normales Zeichen — ‖ — 75
(äquivalent mit `\Vert`)
(In der Kombination mit `\left`,`\right` und `\big..` ist dieses wachsend; als `\parallel` ist es eine Relation.)
D `\def\|{\delimiter "26B30D }`

`\` *escape, backslash:* Einleitungszeichen für TEX-Befehle

`<` Mathematiksatz: Relation `$1<2$` ergibt $1 < 2$

`=` Mathematiksatz: Relation `$0=1-1$` ergibt $0 = 1 - 1$

`\=` Textsatz: `\=o` liefert — ō — 25
D `\def\=#1{{\accent22 #1}}`

`>` Mathematiksatz: Relation `$2>1$` ergibt $2 > 1$

`\>` Mathematiksatz: 78
mittlerer mathematischer Leerplatz vom Umfang ||.
Dies entspricht dem Umfang von `\medmuskip`.
D `\def\>{\mskip \medmuskip }`

`\,` Mathematiksatz: 78
kleiner mathematischer Leerplatz vom Umfang |
Dies entspricht dem Umfang von `\thinmuskip`.
D `\def\,{\mskip \thinmuskip }`

`.` Satzzeichen (Punkt). 16
Nach einem Satzzeichen wird etwas mehr Leerplatz gelassen, es sei denn, es ist "`\frenchspacing`" eingestellt.

`\.` Textsatz: (Punktakzent) `\.o` liefert — ȯ — 25
D `\def\.#1{{\accent95 #1}}`

`;` Satzzeichen (Semikolon)

`\;` Mathematiksatz: 78
großer mathematischer Leerplatz vom Umfang ||
Dies entspricht dem Umfang von `\thickmuskip`.
D `\def\;{\mskip \thickmuskip }`

`?` Satzzeichen (Fragezeichen)

`?‘` Satzzeichen (Fragezeichen, Akzent) *Ligatur* — ¿ —
`?‘De verdad?` liefert — ¿De verdad? —
Dieser Befehl ist abhängig von der aktuellen Schrift. Standardmäßig ist diese Ligatur in den Computer Modern Fonts enthalten.

`!` Satzzeichen (Ausrufezeichen)

`!‘` Satzzeichen (Ausrufezeichen, Akzent) *Ligatur* — ¡ —
`!‘No me digas!` liefert — ¡No me digas! —
Dieser Befehl ist abhängig von der aktuellen Schrift. Standardmäßig ist diese Ligatur in den Computer Modern fonts enthalten.

`\!` Mathematiksatz: negativer kleiner Leerplatz *(backskip)*, 78
der dem Umfang von `-\thinmuskip` entspricht.
`$\int\int_a^b f(x) = \int\!\!\int_a^b f(x)$`
ergibt $\int \int_a^b f(x) = \int\!\!\int_a^b f(x)$
D `\def\!{\mskip -\thinmuskip }`

`_` *Unterstrich* Mathematiksatz: Einleitungsbefehl für Indizes, für alles, was “unten steht”. 65
`$x_{ij}$` liefert x_{ij}
Ersatzbefehl mit gleicher Wirkung: `\sb` (subscript)

`\_` liefert im Textsatz — _ — 15
(konstruiert durch einen `\hrule`-Befehl !)
D
```
\def\_{\leavevmode \kern .06em
       \vbox {\hrule width.3em}}
```

`^` *Dach, Zirkumflex* Einleitungsbefehl im Mathematiksatz für Exponenten, für alles, was “oben steht”. 65
`$x^2$` liefert x^2
Ersatzbefehl mit gleicher Wirkung: `\sp` (superscript)

`\^` Textsatz: (Zirkumflex) `\^a` liefert — â — 25
D `\def\^#1{{\accent94 #1}}`

* `^^` Einleitung für die Ersatzdarstellung der ASCII-Zeichen.
Es werden dabei zwei Fälle unterscheiden:

1. 3 Folgen zwei hexadezimale Ziffern `0`...`9`, `a`...`f` auf `^^`, wie zum Beispiel in `^^4f` ≡ ‘O’, dann wird dieser Wert als hexadezimale Ersatzdarstellung für das entsprechende ASCII-Zeichen aus dem 256-er ASCII-Code betrachtet.

2. Es folgt *ein* Zeichen "`@`" bis "`~`", `^^` wirkt wie eine 'control'-Taste. Technisch gesehen wird vom ASCII-Codewert des nachfolgenden Zeichens 64 subtrahiert und das so entstandene neue Zeichen eingesetzt.
   `^^M` steht etwa für *control M = CR = return*. Aus `^^a` wird '!'. Hat das folgende Zeichen einen ASCII-Wert, der kleiner als 64 ist, so wird statt dessen 64 addiert. Aus `^^0^^!` wird 'pa'.

`~` *Tilde*, gibt ein Leerzeichen aus und verhindert die Trennung an dieser Position (geschützter Leerschritt). 29
Beispiel: `Dr.~A.~B.~Meier`

```
D \catcode`\~=\active
  \def~{\penalty\@M \ }
```

`\~` Textsatz: (Tilde) `\~ o` liefert — õ — 25

```
D \def\~#1{{\accent "7E #1}}
```

`@` *at-Zeichen (commercial at,'Klammeraffe')* Textzeichen, das von einigen *plain-TEX*-Makros als Sonderzeichen benutzt wird, um Befehlsnamen zu verstecken. Dies sind im Abschnitt 14.2 beschrieben.

`\aa` Textsatz: liefert — å— 25, 25

```
D \def\aa{\accent 23a}
```

`\AA` Textsatz: liefert — Å— 25, 25

```
  \def\AA{\leavevmode \setbox 0\hbox {h}\dimen@ \ht 0
D       \advance \dimen@ -1ex\rlap {\raise .67\dimen@
            \hbox {\char'27}}A}
```

∗ `\above` Mathematiksatz: Bruch mit vorgebbarer Dicke des einzusetzenden Bruchstriches 72

`$$ a \above 2pt b $$` liefert $\frac{a}{b}$
(siehe auch `\atop`, `\over`)

∗ `\abovedisplayshortskip`
Leerraum über einer mathematischen Formel im *display-style*, falls die letzte Zeile des vorangehenden Absatzes so kurz ist, daß sie nicht in den Formelteil hineinragt.
Vorbesetzt: `\abovedisplayshortskip=0pt plus 3pt`

∗ `\abovedisplayskip`
Leerraum über einer mathematischen Formel im *display-style* (siehe auch `\abovedisplayshortskip`).
Vorbesetzt:
`\abovedisplayskip=12 pt plus 3pt minus 9pt`

∗ `\abovewithdelims` 77
Mathematiksatz: Bruch mit vorgebbarer Dicke des einzusetzenden Bruchstriches und der verwendeten Klammern.
`$$ a \abovewithdelims<> 2pt b $$` liefert $\left\langle \frac{a}{b} \right\rangle$
(siehe auch `\atopwithdelims`, `\overwithdelims`)

∗ `\accent` Textsatz: TEX-Primitivbefehl zur Bildung von Akzenten im Textsatz. Zum Beispiel ist `\"` definiert durch `\def\"#1{{\accent"7F #1}}` (Auf Position `"7F` (=127) der Computer Modern Fonttabellen stehen die Trema (Pünktchen) für den Umlaut.) 25, *52*

`\active` steht als Benennung des `\catcode`-Wertes für *aktive* Zeichen. Dies sind solche, die selbst wieder in einer Definition verwendet werden können. `~` (Tilde) ist ein aktives Zeichen. *26*, *185*
Beispiel: `\catcode‘\~=\active`
D `\chardef\active=13`

`\acute` Mathematiksatz: mathematischer Akzent 69
`$\acute x$` ergibt $\acute{x}$
D `\def\acute{\mathaccent "7013 }`

∗ `\adjdemerits` Minuspunkte, die beim Absatzumbruch für zwei “visuell inkompatible” Zeilen aufgerechnet werden. Dies sind zwei aufeinanderfolgende Zeilen, bei denen die eine mit extra viel Leerraum durchschossen wurde und die andere mit wenig.
Vorbesetzt: `\adjdemerits=10000`

∗ `\advance` Allgemeiner Additionsbefehl — zum Beispiel: *30*, *37*, *46*
```
\advance\baselineskip by -1pt
\advance\leftskip by 1cm
\advance\count7 by -3
\advance\pageno by 2
```
(siehe auch `\multiply`, `\divide`, `\subtract`)

`\advancepageno` ist ein internes Hilfsmakro der *output-routine* zum Fortschalten der Seitenzählung. Ist die Seitennummer größer als Null, wird sie um 1 erhöht, sonst um 1 erniedrigt. 174
(Die Ausgabe erfolgt über `\folio`.)
```
D \def\advancepageno{\ifnum \pageno <\z@ \global
        \advance \pageno \m@ne \else
          \global \advance \pageno \@ne \fi }
```

`\ae` Textsatz: skandinavische æ-Ligatur. 25, 25
Die Position (`"1A`) ist hier durch die Computer Modern Schriften festgelegt.
D `\chardef\ae="1A`

`\AE` Textsatz: skandinavische Æ-Ligatur. 25, 25
Die Position (`"1D`) ist hier durch die Computer Modern Schriften festgelegt.
D `\chardef\AE="1D`

∗ `\afterassignment` 119, *120*
Genau ein Zeichen (token) oder ein Befehl, der als Parameter folgt, wird zwischengespeichert und erst *nach* der Abarbeitung der *nächsten Zuweisung* ausgeführt.
Beispiel:
`\afterassignment\abc ... \dimen1=10pt•...`
`\afterassignment\abc ... \setbox0=\hbox{•...}`
Jeweils an der durch "•" markierten Stelle, wird der Befehl `\abc` eingefügt. Bei `\afterassignment` wird ein globales Register verwendet, jeweils der letzte Aufruf ist hinterher gültig.

∗ `\aftergroup` Das nächste *token* oder der nächste Befehl wird gespeichert und erst nach dem Schließen des aktuellen Klammerniveaus abgearbeitet. Mehrere `\aftergroup` Befehle sammeln die gespeicherten Befehle auf.
`{\aftergroup\eins\aftergroup\zwei ...}•...`
An der durch "•" markierten Stelle wird `\eins\zwei` eingefügt.

`\aleph` Mathematiksatz: liefert das Symbol — ℵ — 88
D `\mathchardef\aleph="240`

`\allocationnumber`
interner Zähler für die `\new...`-Befehle.
`\newcount`, `\newdimen` ...
Darf nicht extern verändert werden !
D `\countdef\allocationnumber=21`

`\allowbreak` Mathematikmodus: setzt für Formeln im *text-style* eine erlaubte Trennstelle für den Zeilenumbruch. 89
D `\def\allowbreak{\penalty \z@ }`

`\alpha` Mathematiksatz: griechischer Buchstabe — α — 66
D `\mathchardef\alpha="10B`

`\amalg` Mathematiksatz: binärer Operator — ∐ — 86
D `\mathchardef\amalg="2271`

`\angle` Mathematiksatz: (konstruiertes) Symbol — ∠ — 88
```
D \def\angle{{\vbox {\ialign {$\m@th
      \scriptstyle ##$\crcr
      \not \mathrel {\mkern 14mu}\crcr \noalign
      {\nointerlineskip } \mkern 2.5mu\leaders \hrule
      height.34pt\hfill \mkern 2.5mu\crcr }}}}
```

`\approx` Mathematiksatz: Relation — ≈ — 86
D `\mathchardef\approx="3219`

`\arccos` Mathematiksatz: Operator — arccos — 78
D `\def\arccos{\mathop {\rm arccos}\nolimits }`

`\arcsin` Mathematiksatz: Operator — arcsin — 78
D `\def\arcsin{\mathop {\rm arcsin}\nolimits }`

`\arctan` Mathematiksatz: Operator — arctan — 78
D `\def\arctan{\mathop {\rm arctan}\nolimits }`

`\arg` Mathematiksatz: Operator — arg — 78
D `\def\arg{\mathop {\rm arg}\nolimits }`

`\arrowvert` Mathematiksatz: Klammer, Delimiter nur mit `\left`, `\right` oder `\big...` verwendbar. 75
`$\big\arrowvert$` liefert $\big\arrowvert$
D `\def\arrowvert{\delimiter "33C000 }`

`\Arrowvert` Mathematiksatz: Delimiter nur mit `\left`, `\right` oder `\big..` verwendbar. 75
`$\big\Arrowvert$` liefert $\big\Arrowvert$
D `\def\Arrowvert{\delimiter "33D000 }`

`\ast` Mathematiksatz: binärer Operator 86
`$a\ast b$` liefert — $a \ast b$ —
D `\mathchardef\ast="2203`

`\asymp` Mathematiksatz: Relation — $\asymp$ — 86
D `\mathchardef\asymp="3210`

`at` Schlüsselwort für `\font`-Befehle. 57
Beispiel: `\font\bigrm=cmr10 at 12pt`

∗ `\atop` Mathematiksatz: setzt Bruch ohne Bruchstrich. 72, *74*, *82*
Beispiel: `$$ x \atop x+1 $$` liefert $$x \atop x+1$$
(siehe auch `\above`, `\over`)

∗ `\atopwithdelims` 77
Mathematikmodus: setzt Bruch ohne Bruchstrich mit vorgebbaren, aber automatisch wachsenden Klammern.
Beispiel: `$ n \atopwithdelims<> k$` liefert $\left\langle {n \atopwithdelims<> k} \right\rangle$
(siehe auch `\abovewithdelims`, `\overwithdelims`).

`\b` Textsatz: *bar-under* Akzent `\b o` liefert — o̲ — 25
D
```
\def\b#1{\oalign {#1\crcr \hidewidth
    \vbox to.2ex{\hbox {\char 22}\vss }\hidewidth }}
```

`\backslash` Mathematiksatz: normales Zeichen. 75, 88
`$ M\backslash N $` liefert — $M\backslash N$ —
(siehe auch: binärer Operator `\setminus`).
D `\def\backslash{\delimiter "26E30F }`

3 ∗ `\badness` liefert die Bewertung (badness) der letzten erzeugten Box (`\hbox`, `\vbox`) aus. Eine überfüllte Box, beispielsweise die typische 'overfull hbox', hat den Wert 100000, sonst liegt die Bewertung zwischen 0 und 10000 (siehe auch `\hbadness`, `\vbadness`).

`\bar` Mathematiksatz: Akzent `$\bar x$` liefert — $\bar{x}$ — 70
D `\def\bar{\mathaccent "7016 }`

∗ `\baselineskip` normaler Abstand der Grundlinien der Textzeilen. 30ff, *51*, 145
Vorbesetzt: `\baselineskip=12pt`
Dabei hat die Zeile eine Höhe von 8.5 pt und eine Tiefe (für Unterlängen) von 3.5 pt. Durch Veränderung dieses Wertes können die Zeilen mit unterschiedlichem Abstand (Durchschuß) gesetzt werden.

∗ `\batchmode` stellt den *batchmode* ein: Im Fehlerfall wird *keine* vorrangige Anfrage gestellt, sondern Fehler werden nur noch in der Protokolldatei verzeichnet. 164

∗ `\begingroup` erzeugt eine neue Blockstruktur, ähnlich wie "`{`". Dieser Block muß allerdings durch ein korrespondierendes `\endgroup` geschlossen werden. *35*

∗ `\beginsection` ist eigentlich ein Beispielmakro aus plain-TEX zur Illustration, wie ein Abschnitt mit zusätzlichem Leerraum davor und dahinter sowie einer Hervorhebung als linksbündiger Zeile beginnen kann. Zusätzlich wird der Titel des Abschnitts protokolliert. Der Aufruf hat die Form `\beginsection` *Titel* `\par`. Dabei sollte der *Titel* nicht länger als eine Zeile sein, da er als einfache linksbündige Zeile gesetzt wird.

D
```
\outer\def\beginsection#1\par{%
            \vskip\z@ plus.3\vsize\penalty-250
            \vskip\z@ plus-.3\vsize
            \bigskip
            \vskip\parskip
            \message{#1}%
            \leftline{\bf#1}%
            \nobreak\smallskip\noindent}
```

∗ `\belowdisplayshortskip`
Mathematiksatz: gibt den Leerraum unter einer hervorgehobenen Formel im *display-style* an, falls die folgende Absatzzeile die Formel nicht überlappt.
Vorbesetzt:
`\belowdisplayshortskip=7pt plus 3pt minus 4pt`

∗ `\belowdisplayskip`
Mathematiksatz: Leerraum unter einer hervorgehobenen Formel im *display-style*.
Vorbesetzt:
`\belowdisplayskip=12pt plus 3 pt minus 9pt`

`\beta` Mathematiksatz: griechischer Buchstabe — β — 66
D `\mathchardef\beta="10C`

`\bf` **boldface** Schrift(-familie) anwählen. 56
D `\def\bf{\fam \bffam \tenbf }`

`\bffam` interne Nummer (6) der **boldface**-Schriftfamilie. (Nicht ändern!)

```
D \newfam\bffam
  \def\bf{\fam\bffam\tenbf}
```

`\bgroup` wirkt wie ein "`{`" — damit wird eine neue Klammergruppe eröffnet. Diese kann mit "`}`" oder "`\egroup`" wieder geschlossen werden. Für Makros, die eine geöffnete Gruppe hinterlassen wollen, ist dieser Befehl wichtig.
Nach `\def\test{\bgroup}` wirkt `\test` wie ein " `{` " , durch `\def\test{\hbox\bgroup}` wirkt `\test` wie ein " `\hbox{`".
Allerdings sind die Befehlsfolgen "`\def\test\bgroup`" und "`\def\test{`" verschieden!

`\big` Mathematiksatz: vergrößert den folgenden Begrenzer. Das Ergebnis ist ein normales Zeichen: z.B. `$(a+b) \big\vert (c+d)$` liefert $(a+b)\big|(c+d)$ Das Satzverhalten ist wie bei normalen Zeichen. 76

```
D \def\big#1{{\hbox {$\left #1\vbox to8.5\p@ {}\right
        .\n@space $}}}
```

`\Big` Mathematiksatz: vergrößert den folgenden Begrenzer. `\Big` ist etwa 1.5 × `\big` Das Ergebnis wird wie ein normales Zeichen gesetzt. 76

```
D \def\Big#1{{\hbox {$\left #1\vbox to11.5\p@ {}\right
        .\n@space $}}}
```

`\bigbreak` Textsatz: markiert eine gute Umbruchstelle, gleichzeitig wird so viel Platz wie bei `\bigskip` gelassen. Dabei verhalten sich `\smallbreak`,`\medbreak`, `\bigbreak` wie 1 : 2 : 4. 46

```
D \def\bigbreak{\par \ifdim \lastskip <\bigskipamount
        \removelastskip \penalty -200\bigskip \fi }
```

`\bigcap` Mathematiksatz: großer Operator — $\bigcap$ — 73

```
D \mathchardef\bigcap="1354
```

`\bigcirc` Mathematiksatz: binärer Operator — $\bigcirc$ — 86

```
D \mathchardef\bigcirc="220D
```

`\bigcup` Mathematiksatz: großer Operator — $\bigcup$ — 73

```
D \mathchardef\bigcup="1353
```

`\bigg` Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2 × `\big`. Das Ergebnis wird wie ein normales Zeichen gesetzt. 76

```
D \def\bigg#1{{\hbox {$\left #1\vbox to14.5\p@ {}\right
        .\n@space $}}}
```

\Bigg — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2,5 × \big. (Dies ist die größte Form der '\big.'-Serie.) Das Ergebnis wird wie ein normales Zeichen gesetzt. 76

D
```
\def\Bigg#1{{\hbox {$\left #1\vbox to17.5\p@ {}\right
      .\n@space $}}}
```

\biggl — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2 × \bigl. Das Ergebnis wird wie eine linke Klammer gesetzt. *"big left"* 76

D `\def\biggl{\mathopen \bigg }`

\Biggl — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2,5 × \bigl. Das Ergebnis wird wie eine linke Klammer gesetzt. *"big left"* 76

D `\def\Biggl{\mathopen \Bigg }`

\biggm — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2 × \bigm. Das Ergebnis wird wie eine Relation gesetzt. *"big middle"* 76

D `\def\biggm{\mathrel \bigg }`

\Biggm — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2,5 × \bigm. Das Ergebnis wird wie eine Relation gesetzt. *"big middle"* 76

D `\def\Biggm{\mathrel \Bigg }`

\biggr — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2 × \bigr. Das Ergebnis wird wie eine rechte Klammer gesetzt. *"big right"* 76

D `\def\biggr{\mathclose \bigg }`

\Biggr — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 2,5 × \bigr. Das Ergebnis wird wie eine rechte Klammer gesetzt. *"big right"* 76

D `\def\Biggr{\mathclose \Bigg }`

\bigl — Mathematiksatz: vergrößert den folgenden Delimiter ein wenig. Das Ergebnis wird wie eine linke Klammer gesetzt. *"big left"* 76, *82*

D `\def\bigl{\mathopen \big }`

\Bigl — Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 1,5 × \bigl. Das Ergebnis wird wie eine linke Klammer gesetzt. *"big left"* 76

D `\def\Bigl{\mathopen \Big }`

\bigm — Mathematiksatz: vergrößert den folgenden Delimiter ein wenig. Das Ergebnis wird wie eine Relation gesetzt. *"big middle"* 76

D `\def\bigm{\mathrel \big }`

`\Bigm` Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 1,5 × `\bigr`. Das Ergebnis wird wie eine Relation gesetzt. *"big middle"* 76
D `\def\Bigm{\mathrel \Big }`

`\bigodot` Mathematiksatz: großer Operator — ⊙ — 73
D `\mathchardef\bigodot="134A`

`\bigoplus` Mathematiksatz: großer Operator — ⊕ — 73
D `\mathchardef\bigoplus="134C`

`\bigotimes` Mathematiksatz: großer Operator — ⊗ — 73
D `\mathchardef\bigotimes="134E`

`\bigr` Mathematiksatz: vergrößert den folgenden Delimiter ein wenig. Das Ergebnis wird wie eine rechte Klammer gesetzt. *"big right"* 76, *82*
D `\def\bigr{\mathclose \big }`

`\Bigr` Mathematiksatz: vergrößert den folgenden Delimiter auf den Wert von 1,5 × `\bigr`. Das Ergebnis wird wie eine rechte Klammer gesetzt. *"big right"* 76
D `\def\Bigr{\mathclose \Big }`

`\bigskip` Textsatz: erzeugt vertikalen Leerraum im Betrag von `\bigskipamount`. Dies entspricht 2 × `\medskip` oder 4 × `\smallskip`. Im Standard gilt: `\bigskip` entspricht einer Leerzeile. 27
D `\def\bigskip{\vskip \bigskipamount }`

`\bigskipamount` Textsatz: vertikaler Leerraum, der durch `\bigskip` gesetzt wird.
D
```
\newskip\bigskipamount
\bigskipamount=12pt plus 4pt minus 4pt
```

`\bigsqcup` Mathematiksatz: großer Operator — ⊔ — 73
D `\mathchardef\bigsqcup="1346`

`\bigtriangledown` 86
Mathematiksatz: binärer Operator — ▽ —
D `\mathchardef\bigtriangledown="2235`

`\bigtriangleup` Mathematiksatz: binärer Operator — △ — 86
D `\mathchardef\bigtriangleup="2234`

`\biguplus` Mathematiksatz: großer Operator — ⊎ — 73
D `\mathchardef\biguplus="1355`

`\bigvee` Mathematiksatz: großer Operator — ⋁ — 73
D `\mathchardef\bigvee="1357`

`\bigwedge` Mathematiksatz: großer Operator — ⋀ — 73
D `\mathchardef\bigwedge="1356`

∗ `\binoppenalty` Mathematiksatz: Minuspunkte, die es für das Trennen einer Formel im *text-style* nach einem binären Operator gibt (siehe auch `\relpenalty`).
Vorbesetzt: `\binoppenalty=700`

`\bmod` Mathematiksatz: Operator "mod" als binärer Operator (siehe auch `\pmod`). Beispiel:
`$ 17 \bmod 7 = 3$` liefert — $17 \bmod 7 = 3$ —

D
```
\def\bmod{\mskip -\medmuskip \mkern 5mu \mathbin {\rm
       mod}\penalty 900\mkern 5mu\mskip -\medmuskip }
```

`\bordermatrix` Mathematiksatz: Abwandlung von `\pmatrix` — liefert eine Matrix mit Klammern an anderen Positionen. Die erste Zeile und die erste Spalte werden nicht miteingeklammert, sie gelten als Beschriftung.
`$\bordermatrix{a&b&c\cr d&e&f\cr g&h&i\cr}$`

$$\begin{array}{cc} a & b \quad c \\ d & \left(\begin{array}{cc} e & f \\ h & i \end{array}\right) \\ g & \end{array}$$

```
\def\bordermatrix#1{\begingroup \m@th
  \setbox\z@\vbox{\def\cr{\crcr
        \noalign{\kern2\p@\global\let\cr\endline}}%
    \ialign{$##$\hfil\kern2\p@\kern\p@renwd
      &\thinspace\hfil$##$\hfil
      &&\quad\hfil$##$\hfil\crcr
      \omit\strut\hfil\crcr
         \noalign{\kern-\baselineskip}%
      #1\crcr\omit\strut\cr}}%
D  \setbox\tw@\vbox{\unvcopy\z@%
                    \global\setbox\@ne\lastbox}%
  \setbox\tw@\hbox{\unhbox\@ne\unskip
        \global\setbox\@ne\lastbox}%
  \setbox\tw@\hbox{$\kern\wd\@ne
        \kern-\p@renwd\left(\kern-\wd\@ne
        \global\setbox\@ne\vbox{\box\@ne\kern2\p@}%
    \vcenter{\kern-\ht\@ne
        \unvbox\z@\kern-\baselineskip}\,\right)$}%
  \null\;\vbox{\kern\ht\@ne\box\tw@}\endgroup}
```

`\bot` Mathematiksatz: binärer Operator — $\bot$ — (siehe auch `\top` — $\top$ —) 88

D `\mathchardef\bot="23F`

∗ `\botmark` Abfrage des mittels `\mark` als Markierung gesetzten Textes, und zwar wird die *letzte* Markierung auf der aktuellen Seite ausgeliefert. Dies ist nur innerhalb der *output-routine* sinnvoll, zum Beispiel in `\headline` und `\footline` 178
(siehe auch `\firstmark`,`\topmark`,`\mark`).

`\bowtie` Mathematiksatz: Relation — $\bowtie$ — 86

D
```
\def\bowtie{\mathrel \triangleright \joinrel \mathrel
       \triangleleft }
```

* `\box` gibt den Inhalt eines der 256 Box-Register aus. Der Befehl wird gefolgt von der Nummer eines Box-Registers oder der mittels `\newbox` benannten Nummer. Die so ausgegebene Box ist anschließend leer. Der Befehl wirkt global, das heißt, auch beispielsweise nach `{\box0}` ist die Box `0` in jedem Fall leer. Soll der Inhalt erhalten bleiben, so ist der `\copy` Befehl zu verwenden. 133
`\setbox0=\hbox{abc}`
`\box0` hat die Wirkung wie `\hbox{abc}`
(siehe `\copy`, `\unhbox`, `\unhcopy`, `\unvbox`, `\unvcopy`).

* `\boxmaxdepth` ist die maximale Tiefe *(depth)* einer `\vbox`, die diese annehmen darf. Dieser Wert ist mit `\maxdimen`, das heißt 16383,99999 pt, vorbelegt. Damit dürfen Boxen beliebige Unterlängen ('depth') besitzen. Verwendet wird dies beispielsweise beim Fußnotensatz in der *output-Routine*. Würde der so definierte Maximalwert während der Boxkonstruktion überschritten, versetzt TEX die Grundlinie der Box nach unten. Zum Beispiel:
```
\vbox{\hbox{ganz normal, }}
{\boxmaxdepth=0pt\vbox{\hbox{gar nicht}}}
```
liefert ganz normal, gar nicht

`bp` Maßeinheit *(big point)* 22
72 bp = 1 in = 2,54 cm = 1 Zoll

`\brace` Mathematiksatz: liefert 'Binomialkoeffizienten' mit geschweiften Klammern 72
`$ n \brace k+1$` ergibt $\left\{ {n \atop k+1} \right\}$
definiert durch
`\def\brace{\atopwithdelims\{\}}`
D `\def\brace{\atopwithdelims \{\}}`

`\braceld` Mathematiksatz: Endstück einer geschweiften Klammer *(brace left/down)* — ⎩ —
D `\mathchardef\braceld="37A`

`\bracelu` Mathematiksatz: Endstück einer geschweiften Klammer *(brace left/up)* — ⎧ —
D `\mathchardef\bracelu="37C`

`\bracerd` Mathematiksatz: Endstück einer geschweiften Klammer *(brace right/down)* — ⎭ —
D `\mathchardef\bracerd="37B`

`\braceru` Mathematiksatz: Endstück einer geschweiften Klammer *(brace right/up)* — ⎫ —
D `\mathchardef\braceru="37D`

`\bracevert` Mathematiksatz: dicker senkrechter Strich, entstanden aus einer runden Klammer: Er ist nur mit `\big..` oder `\left`, `\right` verwendbar. `$\big\bracevert$` ergibt — ‖ — 75

D `\def\bracevert{\delimiter "33E000 }`

`\brack` Mathematiksatz: eckige "Binomialform". `$ n \brack k+1 $` ergibt $\left[\begin{smallmatrix} n \\ k+1 \end{smallmatrix}\right]$ 72

D `\def\brack{\atopwithdelims []}`

`\break` Textsatz: erzwingt einen Zeilen- oder Seitenwechsel je nachdem, ob es im Absatz oder zwischen Absätzen steht, durch Absetzen von 10000 Pluspunkten (siehe auch `\nobreak`).

D `\def\break{\penalty -\@M }`

`\breve` Mathematiksatz: mathematischer Akzent. `$\breve x$` gibt — $\breve{x}$ — 70

D `\def\breve{\mathaccent "7015 }`

∗ `\brokenpenalty` Minuspunkte *(penalty)*, die während des Seitenumbruchs für einen Seitenwechsel angerechnet werden, bei dem die letzte Zeile einer Seite mit einer Trennung endet. Vorbesetzt: `\brokenpenalty=100` 53

`\buildrel` Mathematiksatz: Die Befehlsfolge 87

`\buildrel` *oberer Text* `\over` *Relation*

bildet eine neue mathematische Relation durch Übereinandersetzen von verschiedenen Symbolen: `$\buildrel \alpha\beta \over \longrightarrow$` liefert $\overset{\alpha\beta}{\longrightarrow}$

D
```
\def\buildrel#1\over #2{\mathrel {\mathop {\kern \z@
      #2}\limits ^{#1}}}
```

`\bullet` Mathematiksatz: binärer Operator — • — 86

D `\mathchardef\bullet="220F`

`by` ist ein optionales Schlüsselwort bei den Befehlen: *30*

`\advance` ... `by` ...
`\multiply` ... `by` ...
`\divide` ... `by` ...

`\bye` ist der Endebefehl für das TEX-Programm. 20
Damit werden evtl. Einfügungen wie `\topinsert` und Fußnoten noch ausgegeben. Die letzte noch angefangene Seite wird mit Leerraum aufgefüllt.

D `\outer \def\bye{\par \vfill \supereject \end }`

`\c` Textsatz: *cedille* `\c o` liefert — o̧ — 25

D
```
\def\c#1{\setbox \z@ \hbox {#1}\ifdim \ht \z@ =1ex
      \accent24 #1\else
        {\ooalign {\hidewidth \char 24\hidewidth
                        \crcr \unhbox \z@ }}\fi }
```

`\cal` Mathematiksatz: Erzeugt kalligraphische große Buchstaben: $\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$ Aufruf `$\cal A$` ... 88

D `\def\cal{\fam \tw@ }`

`\cap` Mathematiksatz: binärer Operator — $\cap$ — 86

D `\mathchardef\cap="225C`

`\cases` Mathematiksatz: "Matrix" mit einer geschweiften Klammer links und ohne Klammerangabe rechts, die typischerweise bei Definitionen verwendet wird. Dabei wird in der ersten Spalte im Mathematikmodus und in der zweiten im Textmodus gesetzt. 82

```
$f(x)=\cases{x&f\"ur $x>0$\cr -x&sonst\cr}$
```

$$f(x) = \begin{cases} x & \text{für } x > 0 \\ -x & \text{sonst} \end{cases}$$

D
```
\def\cases#1{\left \{\,\vcenter {\normalbaselines
      \m@th \ialign {$##\hfil $&\quad ##\hfil
                          \crcr #1\crcr}}\right .}
```

* `\catcode` ist ein interner TEX-Befehl, um die Interpretationsweise der einzelnen eingegebenen Zeichen zu setzen. TEX kennt die folgenden 16 verschiedenen Behandlungsweisen eines eingegebenen Zeichens: *26*, *35*, *156*, *185*

n	*Interpretation*	*Belegung*
0	Escapesymbol, Befehlsanfang	`\catcode‘\\=0`
1	linke Klammer	`\catcode‘\{=1`
2	rechte Klammer	`\catcode‘\}=2`
3	Mathematikanfang, -ende	`\catcode‘\$=3`
4	Tabulatorzeichen	`\catcode‘\&=4`
5	Zeilenende in der Eingabe	`\catcode‘\^^M=5`
6	Parametersymbol in Makros	`\catcode‘\#=6`
7	mathematischer Exponent	`\catcode‘\^=7`
8	mathematischer Index	`\catcode‘\_=8`
9	ignorieren	`\catcode‘\^^@=9`
10	Leerzeichen	`\catcode‘\ =10`
11	Buchstaben	`A...Z, a...z`
12	sonstige Zeichen	
13	aktive Zeichen, die als eigenständige Befehle ohne ein vorgestelltes "\" verwendet werden. Beispiel: Tilde " ~ "	
14	Kommentareinleitung	`\catcode‘\%=14`
15	ungültige, unerlaubte Zeichen	

Beispiel: Durch `\catcode‘\\$=0` wird das Zeichen "$" zum Escapesymbol und wirkt wie ein "\". Man beachte: Der `\catcode` eines einmal eingelesenen Zeichens, beispielsweise in einer Makrodefinition oder mittels `\futurelet`, kann nicht mehr verändert werden.

`cc` Maßeinheit aus dem Druckgewerbe: *Cicero*. 22
1 cc = 12 dd ≈ 0,451 cm

`\cdot` Mathematiksatz: binärer Operator — · — 86
D `\mathchardef\cdot="2201`

`\cdotp` Mathematiksatz: Satzzeichen — · —
D `\mathchardef\cdotp="6201`

`\cdots` Mathematiksatz: Unterformel — ··· — 80, *83*
`$a_1\cdot a_2\cdot\;\cdots\;\cdot a_n$`
erzeugt $a_1 \cdot a_2 \cdot \;\cdots\; \cdot a_n$
D `\def\cdots{\mathinner {\cdotp \cdotp \cdotp }}`

`\centering` interne Dimension für den Mathematiksatz zum Zentrieren von Formeln mit `\eqalignno`, `\leqalignno`
D `\newskip\centering`
`\centering=0pt plus 1000pt minus 1000pt`

`\centerline` Textsatz: `\centerline{ ..text.. }` gibt diesen Inhalt als Zeile zentriert aus. Der Befehl ist aber nur außerhalb von Absätzen im vertikalen Modus verwendbar. 17, *28*
D `\def\line{\hbox to\hsize}`
`\def\centerline#1{\line {\hss #1\hss }}`

∗ `\char` Textsatz: Gibt ein Zeichen aus dem aktuell eingestellten Font aus. Auf `\char` folgt die Codenummer des Zeichens (0 . . . 255). Die Angabe kann dezimal, oktal oder hexadezimal erfolgen.
Das Zeichen ‘A’ kann als `\char65` in dezimaler oder als `\char‘81` in oktaler oder als `\char"41` in hexadezimaler Notation angesteuert werden.

∗ `\chardef` Damit wird ein Befehlsname einem Zeichen zugeordnet. *26*, *35*
`\chardef\%="25` ist praktisch äquivalent zu der Anweisung `\def\%{\char"25 }`, jedoch expandiert ein mittels `\chardef` definierter Befehlsname bei der Verwendung in einem mit `\edef` erzeugten Makro nicht weiter, sondern bleibt wie im Falle von `\%` unverändert. Anstelle von `\%` kann auch etwa “`\prozent`” stehen.

`\check` Mathematiksatz: mathematischer Akzent. 69
`$\check x$` ergibt — $\check{x}$ —
D `\def\check{\mathaccent "7014 }`

`\chi` Mathematiksatz: griechischer Buchstabe — χ — 66
D `\mathchardef\chi="11F`

`\choose` Mathematiksatz: Binomialkoeffizienten. 72, *82*
`$ n \choose k $` ergibt $\binom{n}{k}$
D `\def\choose{\atopwithdelims ()}`

`\circ` Mathematiksatz: binärer Operator — ∘ — 86
`$ a \circ b \circ c$` gibt $a \circ b \circ c$
D `\mathchardef\circ="220E`

∗ `\cleaders` *centered leaders*
`\cleaders <objekt> <skip>`
Der Inhalt von `<objekt>`, dies kann eine `\hbox`, `\vbox`, `\vrule` oder `\hrule` sein, wird so häufig wiederholt, wie der folgende `<skip>` angibt. `<skip>` kann ein fester Leerraum `\hskip <dimension>` oder `\vskip <dimension>` oder auch ein dynamischer Leerraum `\hfill` oder `\vfill` sein. Bei den beiden letzten wird soweit aufgefüllt, wie durch die Größe der äußeren Box verlangt wird.
Falls die Box-Serie nicht genau paßt, wird das Gesamtergebnis zentriert gesetzt *(centered).*
(Siehe auch `\leaders`, `\xleaders`)

```
\line{\strut\vrule
      \cleaders\hbox{\tt....+....*}\hfill
      \vrule}                  ⇒
```

....+....*....+....*....+....*....+....*

`\cleartabs` Textsatz: Dieser Befehl ist zwischen `\+` und `\cr` erlaubt, also bei Tabulatoren. Er löscht ab der aktuellen Position alle Tabulatoren. Folgen "`&`"s, so werden damit neue Tabulatorpositionen gesetzt. 95

D
```
\def\cleartabs{\global \setbox \tabsyet \null
      \setbox \tabs \null }
```

∗ `\closein` schließt eine separate Eingabedatei. 194
Als Parameter sind die Zahlen 0 . . . 15 oder eine mittels `\newread` benannte Eingabedateinummer möglich.
Die Verarbeitung geschieht mit `\openin` und `\read`.

∗ `\closeout` schließt eine separate Ausgabedatei. 194
Als Parameter sind die Zahlen 0 . . . 15 oder eine mittels `\newwrite` benannte Ausabedateinummer möglich.
Die Verarbeitung geschieht mit `\openout` und `\write`.

∗ `\clubpenalty` Minuspunkte, die beim Seitenumbruch vergeben werden, falls die erste Zeile eines Absatzes allein auf der vorangehenden Seite verbleibt.
Vorbesetzt: `\clubpenalty=150`

`\clubsuit` Mathematiksatz: normales Zeichen — ♣ — 88
(siehe auch `\diamondsuit`, `\heartsuit`, `\spadesuit`)
D `\mathchardef\clubsuit="27C`

`cm` Maßeinheit Zentimeter, bei Maßangaben verwendbar 22

`cmbx` interner Name *bold extended Font.* 31, *57*, 58

`cmex` interner Name (Dateiname) des Fonts für die großen mathematischen Symbole *(math extensions).* 58

`cmmi` interner Name (Dateiname) für den Font mit den *italic* Zeichen im Mathematiksatz *(math italic)*. 58

`cmr` interner Name (Dateiname) *computer modern roman* des Standardschrifttyps. *31*, *51*, 58

`cmsl` interner Name (Dateiname) *slanted* der schrägen Schrift. 58

`cmsy` interner Name (Dateiname) für den Font mit den kleinen Symbolen für den Mathematiksatz. 58

`cmti` interner Name (Dateiname) *text italic* der italic Schrift. 58

`cmtt` interner Name (Dateiname) *typewriter type* der typewriter-Schrift. Diese ist als einzige äquidistant. 58

`\colon` Mathematiksatz: 152
Doppelpunkt in der Funktion als Satzzeichen — : —
D `\mathchardef\colon="603A`

`\columns` Textsatz: Schlüsselwort beim `\settabs`-Befehl. 91
Beispiel: `\settabs 5 \columns` teilt die Zeile in 5 gleich breite Spalten für `\+ .. & .. \cr` Sequenzen ein.

`\cong` Mathematiksatz: Relation — ≅ — 86
D `\def\cong{\mathrel {\mathpalette \@vereq \sim }}`

`\coprod` Mathematiksatz: großer Operator — ∐ — 73
D `\mathchardef\coprod="1360`

∗ `\copy` Kopieren des Inhalts eines Box-Registers, der alte Inhalt bleibt erhalten. Beim Aufruf folgt die Nummer des Box-Register 0 . . . 255 oder die mittles `\newbox` benannte Nummer. 133
`\setbox0=\hbox{Tor ! }\copy0\copy0\copy0`
liefert — Tor ! Tor ! Tor ! —
(siehe `\box`, `\unhbox`, `\unhcopy`, `\unvbox`, `\unvcopy`)

`\copyright` Symbol — © —
D
```
\def\copyright{{\ooalign {\hfil \raise .07ex
      \hbox {c}\hfil \crcr \mathhexbox 20D}}}
```

`\cos` Mathematiksatz: großer Operator — cos — 78
D `\def\cos{\mathop {\rm cos}\nolimits }`

`\cosh` Mathematiksatz: großer Operator — cosh — 78
D `\def\cosh{\mathop {\rm cosh}\nolimits }`

`\cot` Mathematiksatz: großer Operator — cot — 78
D `\def\cot{\mathop {\rm cot}\nolimits }`

`\coth` Mathematiksatz: großer Operator — coth — 78
D `\def\coth{\mathop {\rm coth}\nolimits }`

∗ \count — Referierung eines von 256 numerischen *integer* Registern. 147
\count0 enthält standardmäßig die Seitenzählung. Es kann auch unter dem Namen \pageno referiert werden.
Durch \count1=17 wird ein neuer Wert zugewiesen, mittels \number\count1 wird die Zahl im Klartext ausgegeben.
Die Register \count1 bis \count9 sind frei.
Durch etwa \newcount\zahl kann man sich ein freies Register besorgen, dies kann etwa durch \zahl=333 besetzt und \number\zahl ausgegeben werden.
\romannumeral\zahl erzeugt "cccxxxiii".

∗ \countdef — definiert einen symbolischen Namen für eins der 256 *count*-Register. So ist etwa in plain-TEX mit dem Befehl \countdef\pageno=0 dem Register \count0 der Name \pageno standardmäßig zugeordnet
(siehe auch \count).
Eine Zuordnung eines neuen und freien Registers zu einem Namen geschieht mit \newcount\meincount. Damit steht \meincount als *integer*-Register zur Verfügung. Durch etwa \meincount=777 können Zuweisungen getätigt werden.

∗ \cr — Beendet eine Tabulatorzeile (\+ ... & ... \cr) oder eine Eingabezeile in \halign, aber auch eine Zeile in \matrix,\eqalign ... (Mathematiksatz). 91ff

∗ \crcr — ein *Not*-\cr, das ein \cr erzwingt, falls nicht direkt \cr oder \noalign folgt. Die Anwendung erfolgt als Notbremse in Makros.

\csc — Mathematiksatz: großer Operator — csc — 78
D \def\csc{\mathop {\rm csc}\nolimits }

∗ \csname — ist nur in der Kombination 126
\csname ... \endcsname
verwendbar. Aus dem Text (!) zwischen diesen beiden Befehlen wird der Name *eines* TEX-Befehls gebildet und dann ausgeführt. Damit lassen sich auch Befehlsnamen zusammensetzen. Beispiel:
\def\duo#1#2{\csname#1#2\endcsname}
\duo{small}{skip} erwirkt \smallskip

\cup — Mathematiksatz: binärer Operator — ∪ — 86
D \mathchardef\cup="225B

\d — Textsatz: *dot under* Akzent \d o gibt — o̤ — 25
D \def\d#1{\oalign {#1\crcr \hidewidth .\hidewidth }}

\dag — Mathematiksatz: normales Zeichen — † — 88
D \def\dag{\mathhexbox 279}

`\dagger` Mathematiksatz: binärer Operator — † — 86
D `\mathchardef\dagger="2279`

`\dashv` Mathematiksatz: Relation — ⊣ — 86
(siehe auch `\vdash` — ⊢ —)
D `\mathchardef\dashv="3261`

∗ `\day` ist intern mit dem Tag des Tagesdatums besetzt, durch `\the\day` oder `\number\day` wird es ausgegeben.
(siehe auch `\month`, `\year`, `\time`)

`dd` Maßeinheit aus dem Druckgewerbe: *Didôt Punkt.* 22
1157 dd = 1238 pt, 1 dd ≈ 0,376 mm

`\ddag` Mathematiksatz: normales Zeichen — ‡ — 88
D `\def\ddag{\mathhexbox 27A}`

`\ddagger` Mathematiksatz: binärer Operator — ‡ — 86
D `\mathchardef\ddagger="227A`

`\ddot` Mathematiksatz: (*double dot*) mathematischer Akzent 69
`$\ddot x$` liefert — $\ddot{x}$ —
D `\def\ddot{\mathaccent "707F }`

`\ddots` Mathematiksatz: (*diagonal dots*) Unterformel — $\ddots$ — 80
(siehe auch `\ldots`, `\vdots`, `\cdots`)
D
```
\def\ddots{\mathinner {\mkern 1mu
    \raise 7\p@ \vbox {\kern7\p@ \hbox {.}}\mkern 2mu
    \raise 4\p@ \hbox{.}\mkern 2mu
    \raise \p@ \hbox {.}\mkern 1mu}}
```

∗ `\deadcycles` internes Zählregister für die *output-routine*, um Endlosschleifen in einer *output-routine* zu verhinden. Damit werden die aufeinander folgenden Aufrufe gezählt, bei denen keine `\shipout`-Ausgabe erfolgt.

∗ `\def` Befehl zur Definition von Makros — zum Beispiel: 109ff
`\def\DoubleBigskip{\bigskip \bigskip}`
Ein Makro mit Parameter wäre:
`\def\zeile#1{#1_1,#1_2,#1_3\ldots#1_n}`
angewendet in `$\vec\gamma=(\zeile\gamma)$`:

$$\vec{\gamma} = (\gamma_1, \gamma_2, \gamma_3 \ldots \gamma_n)$$

∗ `\defaulthyphenchar`
ist ein fontspezifisches Zeichen, das beim Trennen am Zeilenende eingesetzt wird (normalerweise "-"). Es besetzt den fontspezifischen Wert `\hyphenchar`, der beim `\font`-Befehl belegt wird. Dieser kann nachträglich verändert werden. Nach
`\hyphenchar\tenrm="3D` oder
`\hyphenchar\tenrm=‘\=`
wird ein "=" als Trenner im Font `\tenrm` verwendet
(siehe auch `\hyphenchar`).

∗ `\defaultskewchar`
fontspezifisches Zeichen als Referenzsymbol zur Akzentbildung im Mathematikmodus — entsprechend den Dimensionen dieses Zeichens werden dann auch andere Akzente gesetzt.

`\deg` Mathematiksatz: großer Operator — deg — 78
D `\def\deg{\mathop {\rm deg}\nolimits }`

∗ `\delcode` Mathematiksatz (interner Befehl) definiert zusätzliche Information für Eingabezeichen und zwar wird eine kleinere Variante für die Symboldarstellung im *text-style* und eine größere Variante im *display-style* bestimmt. Für die Zeichen `( ) [ ] < > \ / |` ist dies geschehen. Diese Information wird verwendet, wenn ein solches Zeichen einem der Befehle `\left`, `\right`, `\abovewithdelims`, `\atopwithdelims` oder `\overwithdelims` folgt. Die Eingabe wird in kompakter hexadezimaler Form geschrieben. Die eckige Klammer "`[`" ist in folgender Weise definiert:

```
\delcode`[="05B302
```

Die Eingabe besteht aus den beiden Teilen '`0 5B`' sowie '`3 02`', dabei ist jeweils das erste Zeichen die Nummer der Schriftfamilie. '`0`' ist `cmr` für die *text-style* Fassung. '`3`' ist `cmex` für die *display-style* Fassung. Die beiden folgenden Ziffern geben jeweils den Platz in der Codetabelle an.

∗ `\delimiter` Mathematiksatz (interner Befehl): Mit diesem können ähnlich wie bei `\delcode` große und kleine Zeichenvarianten angegeben werden. Zusätzlich wird noch die satztechnische Funktion festgelegt.
`\def\langle{\delimiter"426830A }` definiert den Befehl `\langle`. Dabei zerlegt sich die siebenstellige Angabe in '`4 / 268 / 30A`'. Die erste Stelle gibt an: 'linke Klammer', die beiden folgenden Gruppen werden wie bei `\delcode` als die Positionen für große und kleine Varianten in den Codetabellen interpretiert.

Folgende Satzklassen zur Regelung des Leerraums in mathematischen Formeln kennt TEX:

0 normales Zeichen (α)
1 (großer) Operator ($\sum$)
2 binärer Operator ($\star$)
3 Relation ($>$)

4 öffnende (linke) Klammer ({)
5 schließende (rechte) Klammer (})
6 Satzzeichen (.)
7 (Sonderfunktion: variable Schriftfamilie)
(siehe auch `\mathchar`)

∗ `\delimiterfactor`
Mathematiksatz (interne Größe): gibt an, wie groß eine Klammer im Verhältnis zur Unterformel, die sie begrenzt, mindestens sein muß.
Vorbesetzt: `\delimiterfactor=901`
Das heißt mindestens 90,1 %.
Beispiel:
`$$\delimiterfactor=2500 \left [ \Omega \right ] $$` ergibt $\left[\Omega\right]$

∗ `\delimitershortfall`
Mathematiksatz (interne Größe): gibt die maximale Abweichung einer Klammergröße von der Größe der zu begrenzenden Formel an.
Vorbesetzt: `\delimitershortfall=5pt`

`\delta` Mathematiksatz: griechischer Buchstabe — δ — 66
D `\mathchardef\delta="10E`

`\Delta` Mathematiksatz: griechischer Buchstabe — Δ — 67
D `\mathchardef\Delta="7001`

`depth` Schlüsselwort für `\vrule` und `\hrule`-Befehle zur Angabe der Tiefe (Unterlänge) eines Strichs. Wird die Tiefe eines Striches nicht angegeben, so wird die Länge beim `\vrule`-Befehl durch die umgebende Box bestimmt. Beim `\hrule`-Befehl ist `depth` mit 0 pt vorbelegt. 53
Beispiel: `\vrule height 5cm depth 2cm width0.4pt`
(siehe auch `height` und `width`)

`\det` Mathematiksatz: großer Operator — det — 78
D `\def\det{\mathop {\rm det}}`

`\diamond` Mathematiksatz: binärer Operator — $\diamond$ — 86
D `\mathchardef\diamond="2205`

`\diamondsuit` Mathematiksatz: normales Zeichen — $\diamondsuit$ — 88
(siehe auch `\clubsuit`, `\heartsuit`, `\spadesuit`)
D `\mathchardef\diamondsuit="27D`

`\dim` Mathematiksatz: großer Operator — dim — 78
D `\def\dim{\mathop {\rm dim}\nolimits }`

∗ `\dimen` referiert eins von den Dimensionsregistern 0 . . . 255. *146*, 147
Mit `\dimen0=5cm` läßt sich ein solches Register belegen, mit beispielsweise `\vskip\dimen0` oder `\hbox to \dimen0` abrufen. Mit Hilfe von `\newdimen\meins` wird auf `\meins` die Nummer des nächsten freien Registers abgeliefert. (0 bis 9 sind frei, werden aber zum Teil von einigen plain-TEX-Makros mitbenutzt.)
Dann kann `\meins=5cm` und `\hbox to \meins{...}` geschrieben werden.
`\dimen`-Register enthalten keinen *glue*, das heißt, nur Längenangaben ohne "`plus`"- und "`minus`"-Anteile können verwendet werden. Für die dynamischen Längenangaben werden *skip*-Register im Textsatz und *muskip*-Register im Mathematiksatz verwendet.

∗ `\dimendef` benennt ein Dimensionsregister mit einem Namen: Mit etwa `\dimendef\mydimen=17` kann dann dies in `\hbox to \mydimen{...}` verwendet werden. Besser ist es, sich vorher mit beispielsweise `\newdimen\mydimen` ein freies Register zu reservieren und zu benennen.
Der Befehl `\dimendef` wird intern von `\newdimen` zur Benennung verwendet.

∗ `\discretionary` Angabe einer möglichen Trennstelle. 52
Es wird jeweils der Text vor der Trennung, nach der Trennung und der Text, wenn gar nicht getrennt wird, angegeben.

`\discretionary{` *vor* `}{` *nach* `}{` *ohne* `}`

Der Vortrenner `\-` ist intern bereits als die Befehlsfolge `\discretionary{-}{}{}` definiert.
`\def\ck{\discretionary{k-}{k}{ck}}`
hilft in "`ba\ck en`"

∗ `\displayindent` Einrückung für Formeln im *display-style*.
Dies ist ein internes Register, das durch TEX automatisch zu Beginn jeder Formel nach "`$$`", abhängig von `\hangindent` und insbesondere der Länge der letzten vorangehenden Zeile, besetzt wird.

∗ `\displaylimits` Mathematiksatz: Setzt das *limits*-Verhalten mathematischer Operatoren auf den Standard zurück. Durch den Befehl `\limits` kann dafür gesorgt werden, daß die Index- und Exponententeile stets direkt über das Operatorsymbol gesetzt werden, durch `\nolimits` entsprechend neben die Symbole. Beispiel:
`\def\SUM{\sum\limits}` und `\SUM\displaylimits`

`\displaylines` Mathematiksatz: Makro, um mehrere Formeln untereinander zentriert zu setzen.
`\displaylines{`$formel_1$`\cr` $formel_2$`\cr ...}`

```
D  \def\displaylines#1{\displ@y \halign
       {\hbox to\displaywidth
           {$\@lign \hfil \displaystyle ##\hfil $}\crcr
        #1\crcr}}
```

* `\displaystyle` Mathematiksatz: Zwangsweises Setzen des *display-style* Satzmodus einer hervorgehobenen Formel. Die Formel oder der Formelteil wird dann wie bei einem Einschluß in `$$ ... $$` gesetzt (siehe auch `\textstyle`, `\scriptstyle`, `\scriptscriptstyle`). 70, *70*

* `\displaywidowpenalty`
ist ein internes Register mit den Minuspunkten, die beim Seitenumbruch aufgerechnet werden, falls die letzte Zeile eines Absatzes gerade noch auf die nächste Seite kommt, und zwar in dem Spezialfall, daß direkt danach eine hervorgehobene Formel folgt.
Vorbesetzt: `\displaywidowpenalty=50`

* `\displaywidth` Mathematik (interne Größe): Sie gibt die maximale Breite für eine Formel im *display-style* an. Dieser Wert wird zu Beginn jeder *display-style*-Formel `$$...$$` automatisch neu bestimmt. Dieser Wert ist abhängig von den Größen `\hsize`, `\hangindent` und anderem.

`\div` Mathematiksatz: binärer Operator — ÷ — 86
D `\mathchardef\div="2204`

* `\divide` Allgemeiner Divisionsbefehl für Register, wobei der Divisor eine ganze, auch negative Zahl sein muß. Bei der Division wird stets abgerundet.
Beispiel:
`\divide\vsize by 2`
`\divide\count0 by 3`
`\divide\baselineskip by 3`
Man kann allerdings auch einem Register einen Faktor voranstellen, zum Beispiel `0.5\vsize`.

`\do` wird durch das Makro `\dospecials` aktiviert. Es wird je nach Bedarf mit einer anderen Bedeutung versehen. Vorbelegt ist dieser Befehl mit `\relax`.
Nach der Definition `\def\do#1{\catcode‘#1=\other}` kann mit einem nachfolgenden `\dospecials` die Deaktivierung aller besonderen Befehlszeichen erreicht werden.

`\dospecials` Internes TEX-Makro, das beim Aufruf für alle irgendwie besonderen Zeichen einmal einen Makroaufruf `\do` mit dem jeweiligen Zeichen als Parameter generiert. Dies sind das Leerzeichen sowie `\`, `{`, `}`, `$`, `&`, `%`, `#`, `^`, `_`, `~` sowie `^^K` und `^^A`.

D
```
\def\dospecials{\do \ \do \\\do \{\do \}\do \$\do
   \&\do \#\do \^\do \^^K\do \_\do \^^A\do \%\do \~}
```

`\dosupereject` internes Hilfsmakro der Standard-Output-Routine. Es erzwingt die Ausgabe aller Einfügungen *(insertions)*, die zum Beispiel durch `\footnote` oder `\topinsert` erzeugt wurden, die aber noch nicht ausgegeben sind. 174

D
```
\def\dosupereject{\ifnum \insertpenalties >\z@
      \line{}\kern -\topskip \nobreak
      \vfill \supereject \fi }
```

`\dot` Mathematiksatz: mathematischer Akzent. 69
`$\dot x$` liefert — $\dot{x}$ —

D
```
\def\dot{\mathaccent "705F }
```

`\doteq` Mathematiksatz: Relation — $\doteq$ — 86

D
```
\def\doteq{\buildrel \textstyle .\over =}
```

`\dotfill` Textsatz: füllt eine Box mit Punkten auf. 106
`\hbox to 2cm{A\dotfill Z}` liefert A ..... Z

D
```
\def\dotfill{\cleaders \hbox {$\m@th
      \mkern 1.5mu.\mkern 1.5mu$}\hfill }
```

`\dots` liefert im Text- und Mathematiksatz — ... —. 16

D
```
\def\dots{\relax \ifmmode \ldots
                 \else $\m@th \ldots \,$\fi}
```

∗ `\doublehyphendemerits` 53
Minuspunkte für zwei aufeinanderfolgende Zeilen, wobei in beiden getrennt wird.
Vorbesetzt: `\doublehyphendemerits=10000`

`\downarrow` Mathematiksatz: Relation — $\downarrow$ — 75, 87
wachsend mit `\big..`, `\left` und `\right`

D
```
\def\downarrow{\delimiter "3223379 }
```

`\Downarrow` Mathematiksatz: Relation — $\Downarrow$ — 75, 87
wachsend mit `\big..`, `\left` und `\right`

D
```
\def\Downarrow{\delimiter "322B37F }
```

`\downbracefill` Text-, Tabellensatz: Füllt eine Box in der nötigen Breite mit einer Klammer auf: 106
`\hbox to 3cm{\downbracefill}` ergibt

$\overbrace{\qquad\qquad}$

(siehe auch `\upbracefill`)

D
```
\def\downbracefill{$\m@th \braceld
                   \leaders \vrule \hfill
                   \braceru \bracelu
                   \leaders \vrule \hfill \bracerd $}
```

∗	`\dp`	Gefolgt von einer Box-Registernummer (0 . . . 255) wird damit die Tiefe/Unterlänge *depth* einer Box referiert. Diese kann sowohl abgefragt, als auch in Zuweisungen verändert werden.
		`\setbox0=`
		`  \hbox{\vrule height5cm depth3cm width0.4pt}`
		liefert für `\dp0` die Belegung mit 3 cm
		(siehe auch für *height* `\ht` und `\wd` für *width*).
			134

∗	`\dump`	interner Befehl nur für INITEX zum Abspeichern eines neuen *format-files*.	193
∗	`\edef`	Ein "`\def`" entsprechender Befehl zur Definition von Makros, wobei jedoch der Inhalt des Makros schon bei der Definition soweit wie möglich expandiert wird. `\def\a{ach!} \edef\A{\a\a}` ist äquivalent zu `\def\A{ach!ach!}`. (Hinweis: Ein `\noexpand`, in der Definition vor bestimmte Befehle gesetzt, verhindert deren Expandierung.)	113, *198*
	`\egroup`	Implizites "`}`" — `\egroup` schließt eine Gruppe, die mit `\bgroup` oder einer normalen "`{`"-Klammer eröffnet wurde.	
	`\eject`	Erzwingen eines Seitenwechsels. Meist ist dies nur in der Kombination `\vfill\eject` sinnvoll, da die Seiten zu dem Zeitpunkt des `\eject`-Befehls oft noch nicht vollständig gefüllt sind und der dehnbare Leerraum meist nicht ausreicht. D `\def\eject{\par \break }`	45
	`\ell`	Mathematiksatz: liefert das normale Symbol — ℓ — D `\mathchardef\ell="160`	88
∗	`\else`	gehört zu einer `\if...` oder `\ifcase`-Anweisung und leitet den *else*-Zweig ein. Syntax: `\if...  \else  \fi` Syntax: `\ifcase .. \or .. \or ... \else .. \fi`	*48*, 114ff
	`em`	Maßeinheit: fontspezifische Breite des 'M'. Diese wird durch `\fontdimen6 <`*font*`>` repräsentiert. (Beispiel: `\fontdimen6\tenrm`) Dies ist auch die Breite eines `\quad`: | |.	22
3 ∗	`\emergencystretch`	definiert zusätzlichen dynamischen Leerraum, der falls er von 0 pt verschieden ist, beim Absatzumbruch einen dritten Durchgang bewirkt, wenn bei den beiden ersten Durchläufen kein Umbruch ohne eine "overfull hbox" (zu lange Zeile) gelungen ist. `\emergencystretch` definiert dann den Leerraum, der innerhalb einer Zeile zusätzlich verteilt werden darf.	40

`\empty` ist ein leeres Hilfsmakro zum Vergleich in Abfragen. Zum Beispiel wird mit 116

```
\def\abc#1#2#3{\def\test{#1}%
   \ifx\test\empty    ... für #1 unbesetzt
            \else     ... für #1 belegt
   \fi}
```

im Makro `\abc` geprüft, ob `#1` einen Wert besitzt. Bei einem Aufruf `\abc{}...` bleibt der Parameter `#1` leer. Die beiden Makros `\empty` und `\test` sind dann identisch definiert.

D `\def\empty{}`

`\emptyset` Mathematiksatz: normales Zeichen — ∅ — 88

D `\mathchardef\emptyset="23B`

∗ `\end` ist der Endebefehl des TEX-Programms. Vorzuziehen ist `\bye`, dies impliziert ein `\end`, aber vorher wird noch ein `\vfill` zur Seitenauffüllung gegeben, und evtl. Einfügungen wie Fußnoten, die auf der letzten Seite keinen Platz gefunden haben, werden noch als zusätzliche Seite ausgegeben. 17, 20

∗ `\endcsname` Zweiter Teil der "`\csname ...text... \endcsname`" Kommandofolge. Der durch diese beiden Befehle geklammerte Text (!) wird als Befehl ausgeführt. 126
(siehe auch `\csname`)

`\endgraf` Damit hat man einen "Reservebefehl" für `\par`. Gelegentlich ist es sinnvoll, "`\par`" umzudefinieren.

D `\let\endgraf=\par`

∗ `\endgroup` schließt eine mit "`\begingroup`" begonnene Klammerstruktur, und nur eine solche. *35*
`\begingroup` und `\endgroup` müssen als Paar auftreten.

∗ `\endinput` beendet den Einlesevorgang aus einer Datei. Wird dieser Befehl in einer mit `\input` einzulesenden Datei gefunden, so wird dies als Dateiende interpretiert.

`\endinsert` Zweiter Teil der Befehlsfolgen für Einfügungen:

```
\topinsert  ... \endinsert
\midinsert  ... \endinsert
\pageinsert ... \endinsert
```

49

```
\def\endinsert{\egroup
   \if@mid \dimen@ \ht \z@
     \advance \dimen@ \dp \z@
     \advance \dimen@ 12\p@
     \advance \dimen@ \pagetotal
     \advance \dimen@ -\pageshrink
     \ifdim \dimen@ >\pagegoal
            \@midfalse \p@gefalse \fi
   \fi
   \if@mid \bigskip \box \z@ \bigbreak
   \else
      \insert \topins {\penalty 100
          \splittopskip \z@skip
          \splitmaxdepth \maxdimen
          \floatingpenalty \z@
          \ifp@ge \dimen@ \dp \z@
            \vbox to\vsize {\unvbox \z@
                \kern -\dimen@ }%
          \else \box \z@ \nobreak \bigskip \fi
       }\fi \endgroup }
```

D

`\endline` Damit steht noch ein Befehl für `\cr` zur Verfügung, falls es sich als nötig erweist, `\cr` umzudefinieren.

D `\let\endline=\cr`

* `\endlinechar` Nummer des ASCII-Codes, der das Zeilenende repräsentiert, vorbelegt mit 13 (CR=control M). 195

`\enskip` Leerraum mit der Hälfte eines "`em`" (Druckerviertel) als Breite. Im Gegensatz zu `\enspace` kann an der Leerstelle ein Zeilenwechsel erfolgen. 28

D `\def\enskip{\hskip .5em\relax }`

`\enspace` Leerraum mit der Hälfte eines "`em`" (Druckerviertel) als Breite. Es kann kein Zeilenwechsel an dieser Position erfolgen. 29

D `\def\enspace{\kern .5em }`

`\epsilon` Mathematiksatz: griechischer Buchstabe — ϵ — (siehe auch `\varepsilon` — ε —) 66

D `\mathchardef\epsilon="10F`

`\eqalign` Mathematiksatz: 83
Makro zum Ausrichten von Formeln

```
\eqalign{ erster Teil & zweiter Teil \cr
          erster Teil & zweiter Teil \cr}
```

sorgt dafür, daß die Teile bündig an der durch "`&`" markierten Position untereinander stehen.
Beispiel `$$\eqalign{a&=b+c\cr a-b&=c\cr}$$`

$$\eqalign{a&=b+c\cr a-b&=c\cr}$$

```
\def\eqalign#1{\null \,\vcenter {\openup \jot \m@th
      \ialign {\strut \hfil
          $\displaystyle {##}$&$\displaystyle
              {{}##}$\hfil \crcr #1\crcr }}\,}
```

D

`\eqalignno` Mathematiksatz: Makro zum Ausrichten von Formeln mit rechtsbündiger Numerierung: 85

```
\eqalign{ 1. Teil & 2. Teil  & nr \cr
                        ...
          1. Teil & 2. Teil  & nr \cr}
```

sorgt dafür, daß die Teile bündig an der durch "`&`" markierten Position untereinander stehen.
Beispiel:

```
$$\eqalignno{a&=b+c&(1)\cr a-b&=c&(2)\cr}$$
```

$$a = b + c \tag{1}$$
$$a - b = c \tag{2}$$

D
```
\def\eqalignno#1{\displ@y \tabskip \centering
    \halign to\displaywidth
      {\hfil $\@lign \displaystyle
          {##}$\tabskip \z@skip
       &$\@lign \displaystyle
          {{}##}$\hfil \tabskip \centering
       &\llap {$\@lign ##$}\tabskip \z@skip
       \crcr #1\crcr }}
```

∗ `\eqno` Mathematiksatz: 84
In einer *display-style*-Formel `$$` *formel* `\eqno` *nummer* `$$` verwendet, wird damit eine rechtsbündige Numerierung erzeugt. Das Gegenstück ist `\leqno` für eine linksbündige Formel.
Beispiel: `$$ a^2 + b^2 = c^2 \eqno (1) $$` erzeugt

$$a^2 + b^2 = c^2 \tag{1}$$

`\equiv` Mathematiksatz: Relation — ≡ — 86
D `\mathchardef\equiv="3211`

∗ `\errhelp` besetzt mit `\errhelp{...information...}` einen zusätzlichen Fehlertext, den der Benutzer im Fehlerfall als 'HELP' erfragen kann. Der Fehler und die Ausgabe des dazugehörigen Fehlertextes wird durch den Befehl `\errmessage{..Fehlertext..}` provoziert. Als zusätzlicher Hilfstext kann der letzte Text, der durch ein vorangegangenes `\errhelp` Kommando eingetragen wurde, angefordert werden.

∗ `\errmessage` gibt durch `\errmessage{...Fehlertext..}` eine Fehlermeldung aus. Anschließend ist das Programm in der Fehlerbehandlung. Ein mit `\errhelp` vorgespeicherter Hilfstext, kann in dieser Situation zusätzlich abgefragt werden. *119*, 126

3 ∗ `\errorcontextlines`
ist ein internes Register, das angibt, wie viele Verschachtelungsebenen — typischerweise sind dies verschachtelte Makroaufrufe — maximal bei einer Fehlermeldung protokolliert werden.
D `\errorcontextlines=5`

∗ `\errorstopmode` setzt (wieder) den *error stop mode*, das heißt, in Fehlersituationen wird eine Dialoganfrage gestellt.

∗ `\escapechar` gibt das Zeichen an, das bei der *Ausgabe* von Befehlsnamen als Darstellung für das Escape-Zeichen verwendet werden soll. Bei Angabe eines negativen Wertes wird keinerlei Zeichen für das Escape-Zeichen ausgegeben. 126
Voreingestellt: "\" (ASCII `"5C`)
So liefert `{\escapechar=‘\! \string\bigskip}` den Text "!bigskip".

`\eta` Mathematiksatz: griechischer Buchstabe — η — 66
D `\mathchardef\eta="111`

∗ `\everycr` definiert mit `\everycr{..text..}` eine Befehlsfolge, die automatisch *nach* jedem `\cr`-Befehl ausgeführt wird.

∗ `\everydisplay` definiert mit `\everydisplay{..text..}` eine Befehlsfolge, die automatisch zu Beginn jeder *display-style*-Formel, also direkt nach "`$$`" eingefügt wird. 156

∗ `\everyhbox` definiert mit `\everyhbox{..text..}` eine Befehlsfolge, die automatisch zu Beginn jeder *horizontal box*, das heißt zu Beginn eines `\hbox`-Befehls, ausgeführt wird.

∗ `\everyjob` definiert mit `\everyjob{..text..}` eine Befehlsfolge, die automatisch zu Beginn eines TEX-Laufes ausgeführt wird. Dieser Befehl arbeitet nur mit INITEX, da die Befehlsfolge Bestandteil des *format-files* ist.

∗ `\everymath` definiert mit `\everymath{..text..}` eine Befehlsfolge, die zu Beginn einer Formel im *text-style*, also nach "`$`", ausgeführt wird. 156

∗ `\everypar` definiert mit `\everypar{..text..}` eine Befehlsfolge, die zu Beginn jeden Absatzes automatisch ausgeführt wird. 34, *51*

∗ `\everyvbox` definiert mit `\everyvbox{..text..}` eine Befehlsfolge, die automatisch zu Beginn jeder *vertical box*, das heißt zu Beginn eines `\vbox`-Befehls, ausgeführt wird.

`ex` fontspezifische Maßeinheit, die Höhe des kleinen "x". Dies ist die Höhe der Kleinbuchstaben (Minuskeln) ohne Oberlänge. Der Wert ist gespeichert als `\fontdimen5 <`*font*`>`, z. B. `\fontdimen5\tenrm` 22

∗ `\exhyphenpenalty`
Minuspunkte, die für einen Zeilenwechsel nach einem expliziten Trennstrich, beispielsweise in 'Ein-Ausgabe-Verfahren', aufgerechnet werden. Genau wie es im letzten Satz bei der Eingabe `Ein-Ausgabe-Verfahren` geschehen ist.
Vorbesetzt `\exhyphenpenalty=50`

`\exists` Mathematiksatz: normales Symbol — ∃ — 88
D `\mathchardef\exists="239`

`\exp` Mathematiksatz: großer Operator — exp — 78
D `\def\exp{\mathop {\rm exp}\nolimits }`

∗ `\expandafter` Änderung der Expandierungsreihenfolge: 121, *149*
Nach `\expandafter\eins\zwei` wird zunächst der Befehl `\zwei` expandiert. Dies hat die Auswirkung, daß Makroparameter von `\eins` nur mit Teilen der *Expandierung* von `\zwei` belegt werden können.

∗ `\fam` ist ein internes Register, mit der aktuellen Schriftfamilie für den Mathematiksatz. Es wird zu Beginn jeder Formel automatisch auf −1 zurückgesetzt, wenn der *mathematical mode* beginnt. Es wird durch
`\fam=`n mit $n = 0 \ldots 15$
verändert. Zum Beispiel stellen die Befehle `\rm`, `\it`, `\bf` nicht nur einfach eine Schrift ein, sondern bestimmen mit der Besetzung von `\fam` eine Schriftfamilie. Alle Symbole mit einer *variablen* Schriftfamilie (7) in dem entsprechenden Teilfeld der Deklarationen der Befehle `\delcode`, `\delimiter`, `\mathcode` oder `\mathchar` benutzen dann den aktuellen Wert von `\fam`, falls dieser größer als −1 ist, um den Zeichensatz auszuwählen. Enthält `\fam` etwa den Wert 2, so wird im *text-style* der Zeichensatz verwendet, welcher `\textfont2` zugewiesen ist. Entsprechendes gilt für den *script-style* und den *scriptscript-style* bei der Referierung von `\scriptfont` und `\scriptscriptfont`
(siehe auch `\newfam`).

∗ `\fi` ist der Endebefehle zu einer `\if...` oder `\ifcase`-Anweisung. *48*, 114ff
Syntax: `\if...  .. \else ..  \fi`
Syntax: `\ifcase .. \or .. \or ... \else .. \fi`

`fil` ist eine interne Längeneinheit für dynamisch wachsenden Leerraum erster Stufe. *34*
Die Angabe `\hskip 0pt plus 1fil` entspricht `\hfil`.
Es gilt `fil` ≪ `fill` ≪ `filll`.

`\filbreak` gibt eine mögliche Umbruchstelle für den Seitenwechsel an. Der Text wird, wenn er noch paßt, zwischen zwei `\filbreak`-Befehlen auf die gleiche Seite, sonst auf die folgende Seite gesetzt. 45
D `\def\filbreak{\par \vfil \penalty -200\vfilneg }`

`fill` ist eine interne Längeneinheit für dynamisch wachsenden Leerraum zweiter Stufe. Die Angabe `\hskip 0pt plus 2fill` entspricht `\hfill\hfill`. Es gilt `fil` ≪ `fill` ≪ `filll`.

`filll` ist eine interne Längeneinheit für dynamisch wachsenden Leerraum dritter Stufe. Es gilt `fil` ≪ `fill` ≪ `filll`.

∗ `\finalhyphendemerits` 53
Minuspunkte, die für eine Trennung in der vorletzten Zeile eines Absatzes aufgerechnet werden. Vorbesetzt: `\finalhyphendemerits=5000`

∗ `\firstmark` referiert den mit `\mark` gespeicherten Text, und zwar den zuerst auf der aktuellen Seite eingetragenen. Dies ist meist nur in `\headline`, `\footline`, das heißt während der Output-Routine, sinnvoll (siehe auch `\mark`, `\botmark`, `\topmark`). 178

`\fiverm` Anwahl der 'roman'-Schrift in 5 Punkt Größe. 56
D `\font\fiverm=cmr5`

`\fivebf` Anwahl der 'boldface'-Schrift in **5 Punkt Größe.** 56
D `\font\fivebf=cmbx5`

`\fivei` Anwahl der Mathematikschrift in 5 Punkt Größe; sie wird durch den Mathematiksatz automatisch vollzogen. 58
D `\font\fivei=cmmi5`

`\fivesy` Anwahl der Symbole in 5 Punkt in Exponenten und Indizes zweiter Stufe; sie wird durch den Mathematiksatz automatisch vollzogen. 58
D `\font\fivesy=cmsy5`

`\flat` Mathematiksatz: normales Zeichen — ♭ — (siehe auch `\natural`, `\sharp` — ♮ ♯ —) 88
D `\mathchardef\flat="15B`

∗ `\floatingpenalty`
Internes Register: Anzurechnende Minuspunkte, wenn ein `\insert`-Befehl zur Teilung des Einfügungstextes führt, weil dieser zu lang ist. Bei einigen plain-TEX-Makros wird dieses dynamisch umgesetzt: So setzt der Befehl `\footnote` dieses Register auf 20000.

`\fmtname` Internes Register: Es enthält die Bezeichnung des geladenen *format files*.

`\fmtversion` Internes Register: Es enthält die Version des geladenen *format files*.

`\folio` Ausgabe der aktuellen Seitennummer. 46, *47*, *175*
Ist die Seitennummmer negativ, so werden kleine römische Zahlen "i, ii ,iii ,iv ..." benutzt.

```
\def\folio{\ifnum \pageno <\z@ \romannumeral -\pageno
           \else \number \pageno \fi }
```

∗ `\font` Befehl zur Definition eines Fonts. *51*, 57
Syntax: `\font\`*name*`=`*dateiname*
(evtl. gefolgt von der Skalierung mit `scaled` oder `at`)
Beispiele:

```
\font\ganzfett=cmbx10 scaled \magstep3
\font\fett=cmbx10 at 12pt
\font\san=cmss10
```

∗ `\fontdimen` Referierung eines schriftspezifischen Registers
`\fontdimen` *nummer* `\`*fontname*

Standardparameter für Textfonts

1 *slant*-Faktor zur Positionierung von Akzenten bei geneigten Schriften.
2 Normalabstand zwischen 2 Wörtern, kann durch `\spaceskip` überschrieben werden.
3 Möglicher Zusatzabstand zwischen 2 Wörtern: Dieser kann durch den 'plus-Anteil' von `\spaceskip` überschrieben werden
4 Größe, um die der Abstand zwischen 2 Wörtern wegen des Randausgleichs reduziert werden kann. Sie wird durch den 'minus-Anteil' von `\spaceskip` überschrieben.
5 *x-height*, Höhe der Kleinbuchstaben (Minuskeln), zum Beispiel 'a', 'e' oder 'x', ohne Oberlänge.
Dies ist gleichzeitig die Maßeinheit "`1ex`".
6 *Druckerviertelchen:* Das ist die Breite von 'M'. Dies ist gleichzeitig die Maßeinheit "`1em`" (= `\quad`).
7 Zusätzlicher Leerraum am Satzende, falls nicht gerade `\frenchspacing` eingeschaltet ist.
(Falls `\xspaceskip` verändert wird und von 0 pt verschieden ist, wird dieser Wert für den Leerraum am Satzende verwendet. Die Parameter 2, 3, 4 und 7 werden in diesem Fall nicht benutzt.)

Zusätzliche Parameter im Mathematiksatz für Schriftfamilie 2: `\textfont2`**,** `\scriptfont2` **...**

8...12 Korrekturfaktoren zum Setzen eines Bruchs. Je nach Stil werden Zähler um die Werte 8, 9, 10 und Nenner um 11, 12 nach oben bzw. nach unten verschoben.
13 Exponentpositionierung: Minimalverschiebung im Mathematikmodus nach oben im *display-style*.
14 Exponentpositionierung: Ebenso im 'Reduktionsmodus' in Wurzeln usw., um die Formelhöhe niedrig zu halten.
15 Exponentpositionierung: (kleinster Wert) für die restlichen Fälle.
16 Indexpositionierung: minimaler Abstand des Index von der Grundlinie, falls kein Exponent vorhanden ist.
17 Indexpositionierung: Minimalabstand, falls ein Exponent vorhanden ist. Standardmäßig wird ein Index tiefer gesetzt, wenn über ihn ein Exponent gesetzt wird.
18 minimale Verschiebung für Exponenten.
19 minimale Verschiebung für Indizes.
20 Minimalgröße für Klammern im *display-style*.
21 Das gleiche in sonstigen *styles*.
22 Höhe der waagerechten Symmetrieachse für mathematische Zeichen. (Diese wird auch von `\vcenter` verwendet.)

für Schriftfamilie 3 (Symbolfonts): `\textfont3`...

8 Standarddicke für Bruchstriche etc.
9...13 Setzen von Exponenten und Indizes bei großen Operatoren

Jede Veränderung der `\fontdimen` Parameter ist global. Die Gruppenklammern `{ }` haben keinen Einfluß.

* `\fontname` gibt den Namen eines definierten Fonts aus.
Nach der Definition durch
`\font\tenrm=cmr10` (Standard)
`\font\bigbf=cmbx10 scaled \magstep1`
liefern `\fontname\tenrm` → 'cmr10'
`\fontname\bigbf` → 'cmbx10 at 12pt'

`\footins` interne Nummer des *insertion register* für Fußnoten. Mit diesem arbeitet der `\footnote`-Befehl. 174
D `\newinsert\footins`

`\footline` ist ein *token*-Register in plain-TEX mit dem Inhalt der Seitenfußzeile (siehe auch `\headline` für den Seitenkopf). 38, 47ff, 174

```
D  \newtoks\footline
   \footline={\hss\tenrm\folio\hss}
```

`\footnote` setzt eine Fußnote im Standardformat. 50ff
Aufruf: `\footnote{ symbol }{ text }`

```
D  \def\footnote#1{\let \@sf \empty \ifhmode \edef \@sf
          {\spacefactor \the \spacefactor }\/\fi #1\@sf
          \vfootnote {#1}}
```

`\footnoterule` setzt die Trennungsline bei Fußnoten — wird durch die Output-Routine aufgerufen 50

```
D  \def\footnoterule{\kern -3\p@ \hrule width 2truein
          \kern 2.6\p@ }
```

`\forall` Mathematiksatz: normales Zeichen — ∀ — (siehe auch `\exists` — ∃ —) 88

```
D  \mathchardef\forall="238
```

`\frenchspacing` Textsatz: verändert den Satzmodus für Leerraum nach Satzzeichen. Damit wird nach Punkt, Komma etc. genausoviel Leerraum ausgegeben wie zwischen normalen Wörtern. (Zurückstellung mit `\nonfrenchspacing`) 16

```
D  \def\frenchspacing{\sfcode `\.\@m \sfcode `\?\@m
          \sfcode `\!\@m \sfcode `\:\@m
          \sfcode `\;\@m \sfcode `\,\@m }
```

`\frown` Mathematikatz: Relation — ⌢ — (Gegenstück: `\smile` — ⌣ —) 86

```
D  \mathchardef\frown="315F
```

∗ `\futurelet` führt einen `\let`-Befehl aus, wobei die zugewiesene Information später noch einmal verarbeitet wird. 122
Syntax: `\futurelet\cs` *token*$_1$ *token*$_2$
wirkt wie: `\let\cs=`*token*$_2$ *token*$_1$ *token*$_2$
Damit kann zum Beispiel in

```
\futurelet\next\test\weiter
```

der Inhalt von `\weiter` in `\test` geprüft werden, da dieser bereits in `\next` steht. Das Beispiel ist äquivalent zu den Befehlen `\let\next\weiter\test\weiter`.

`\gamma` Mathematiksatz: griechischer Buchstabe — γ — 66

```
D  \mathchardef\gamma="10D
```

`\Gamma` Mathematiksatz: griechischer Buchstabe — Γ — 67

```
D  \mathchardef\Gamma="7000
```

`\gcd` Mathematiksatz: großer Operator — gcd — 78

```
D  \def\gcd{\mathop {\rm gcd}}
```

∗ `\gdef` äquivalent zu `\global\def` — erzeugt eine globale Definition eines Makros, unabhängig von Blockstrukturen. 114

`\ge` Mathematiksatz: Relation — ≥ — 86
D `\mathchardef\ge="3215`

`\geq` Mathematiksatz: Relation — ≥ — 86
D `\mathchardef\geq="3215`

`\gets` Mathematiksatz: Relation — ← — 87
(äquivalent mit `\leftarrow`)
D `\mathchardef\gets="3220`

`\gg` Mathematiksatz: Relation — ≫ — 86
(siehe auch `\ll` — ≪ —)
D `\mathchardef\gg="321D`

∗ `\global` ist ein Schlüsselwort, das, Zuweisungen oder Makrodefinitionen vorangestellt, die globale Ausführung bewirkt. Damit werden diese unabhängig von der Blockstruktur in allen Klammerebenen gesetzt. 112, *174*

∗ `\globaldefs` TEX-Register, das die Gültigkeitsbereiche für Zuweisungen global steuert:
Vorbesetzt: `\globaldefs=0`
0 (Voreinstellung) Die Zuweisung ist global, wenn ein `\global` davor steht.
< 0 Alle Zuweisungen werden *lokal* ausgeführt, `\global` wird ignoriert.
> 0 Alle Zuweisungen wirken global.

`\goodbreak` markiert eine gute Trennposition im Seitenumbruch. Gleichzeitig geht der Absatz zu Ende. 45
D `\def\goodbreak{\par \penalty -500 }`

`\grave` Mathematiksatz: mathematischer Akzent 69
`$\grave x$` liefert — $\grave{x}$ —
D `\def\grave{\mathaccent "7012 }`

`\H` Textsatz: Doppelakut, *langer ungarischer Umlaut* 25
`\H o` liefert — ő —
D `\def\H#1{{\accent "7D #1}}`

∗ `\halign` Befehl zum Satz von Tabellen *(horizontal alignment)*, wobei die Spalten automatisch auf die nötigen Breiten gebracht werden. 95ff
Syntax:
`\halign{` *musterzeile* `\cr`
eingabezeile$_1$ `\cr`
...
eingabezeile$_n$ `\cr}`

`\hang` Hilfsmakro zum `\item` Befehl. Es setzt `\hangindent` auf den Wert von `\parindent`.
D `\def\hang{\hangindent\parindent}`

∗ `\hangafter` steuert zusammen mit `\hangindent` das Aussehen eines Absatzes. Ist `\hangafter`≥ 0, so werden die ersten `\hangafter` Zeilen um `\hangindent` verkürzt, sonst die darauf folgenden. — 38, 39ff, *45*

∗ `\hangindent` steuert zusammen mit `\hangafter` das Aussehen eines Absatzes. Ist `\hangindent`< 0, so wird der Absatz jeweils links sonst rechts um den Betrag von `\hangindent` verkürzt.
Die Voreinstellung, die nach jedem Absatz automatisch wieder restauriert wird, lautet:
`\hangindent=0pt`, `\hangafter=1` — 38, 39ff, *45*, *51*

`\hat` Mathematiksatz: mathematischer Akzent
`$\hat x$` liefert — $\hat{x}$ —
D `\def\hat{\mathaccent "705E }` — 69

∗ `\hbadness` Grenzwert für schlechte Zeilen, bzw. Boxen, der angibt, ab welcher Negativbewertung eine Box protokolliert wird.
Vorbesetzt: `\hbadness=1000`
(siehe auch `\vbadness`, `\hfuzz`, `\vfuzz`) — 42

`\hbar` Mathematiksatz: normales Zeichen — $\hbar$ —
D `\def\hbar{{\mathchar '26\mkern -9muh}}` — 88

∗ `\hbox` eröffnet eine *horizontal box*.
Dies ist eine Box, in der alles Material nebeneinander gesetzt wird. Eine `\hbox` hat eine natürliche Weite, die durch ihren Inhalt bestimmt ist. Die Weite kann aber auch extern deklariert werden:

```
\hbox to 10cm {...text...}
```

Durch geeigneten dynamischen Leerraum muß sie dann aber auch aufgefüllt werden. Eine zusätzliche Weite, die über die natürliche Weite hinausgeht, wird zum Beispiel durch `\hbox spread 2cm {...text...}` definiert. — *79*, *85*, 130ff, *135*

`\headline` ist das *token*-Register, das in plain-TEX den Inhalt der Seitenkopfzeile bestimmt
(siehe auch `\footline` für die Fußzeile).
D
```
\newtoks\headline
\headline={\hfil}
```
— 38, 47ff, 173

`\heartsuit` Mathematiksatz: normales Zeichen — $\heartsuit$ —
(siehe auch `\spadesuit`, `\clubsuit`, `diamondsuit`)
D `\mathchardef\heartsuit="27E` — 88

`height` Schlüsselwort für `\hrule`, `\vrule`; Befehle zur Angabe der Linienhöhe.
Beispiel: `\vrule height 4cm width0.4pt` — 53

∗ `\hfil` erzeugt wachsenden horizontalen Leerraum erster Stufe. — *34*, 142

∗ `\hfill` erzeugt wachsenden horizontalen Leerraum der zweiten Stufe. Der Befehl dominiert sozusagen ein `\hfil`, das in die andere Richtung zeigt. *35*, *92*, 142

∗ `\hfilneg` entfernt vorangehenden `\hfil` (erster Stufe). 142

∗ `\hfuzz` Grenze, ab der eine "überfüllte `\hbox`" gemeldet wird. Vorbesetzt: `\hfuzz=0.1pt`
(siehe auch `\vfuzz`, `\hbadness`, `\vbadness`) 42, 54

`\hglue` erzeugt horizontalen Leerraum, der auch beim Umbruch nicht entfernt werden kann, indem `\nobreak` und leere `\vrule`s eingefügt werden.
Beispiel: `\hglue 3cm plus 1 cm`
(siehe auch `\topglue`, `\vglue`)

D
```
\def\hglue{\afterassignment \hgl@ \skip@ =}
\def\hgl@{\leavevmode \count@\spacefactor
          \vrule width\z@
          \nobreak\hskip\skip@ \spacefactor\count@}
```

`\hideskip` ist horizontaler Leerraum, der beim Tabellensatz verwendet wird, um ein Element zu setzen, das die logische Breite Null besitzt. Dessen Breite geht praktisch nicht in die Kalkulation der maximalen Spaltenbreite ein.

D
```
\newskip\hideskip
\hideskip=-1000pt plus 1fill
```

`\hidewidth` unterdrückt die Spaltenbreitenberechnung beim Tabellensatz. Genau wird ein großer negativer Rückwärts-Skip verwendet, um eine logische breite von Null zu erzeugen. 103

D
```
\def\hidewidth{\hskip \hideskip }
```

∗ `\hoffset` ist der horizontaler Versatz, mit dem die gesamte Ausgabe bei der Druckausgabe relativ zum Papier nach rechts verschoben werden soll. Vorsicht!
Vorbesetzt: `\hoffset=0pt`
(siehe auch `\voffset`) 37, 38

3 ∗ `\holdinginserts`
ist ein internes Steuerregister. Falls es einen Wert größer als Null enthält, werden vorhandene Einfügungen (*insertions*) während der *output-routine* nicht bereitgestellt, sondern noch gespeichert. Dies ist sinnvoll, wenn innerhalb der *output-routine* der Umbruch noch verändert wird, falls beispielsweise keine Seiten der Länge `\vsize` entstehen sollen, sondern zu mehrspaltigem Satz verarbeitet werden.

D
```
\holdinginserts=0
```

`\hom` Mathematiksatz: großer Operator — hom — 78

D
```
\def\hom{\mathop {\rm hom}\nolimits }
```

`\hookleftarrow` Mathematiksatz: Relation — ↩ — 87
D `\def\hookleftarrow{\leftarrow \joinrel \rhook }`

`\hookrightarrow` 87
Mathematiksatz: Relation — ↪ —
D `\def\hookrightarrow{\lhook \joinrel \rightarrow }`

`\hphantom` setzt die Information in `\hphantom{text}` nur als Leerraum. Die Breite ist wie beim Originaltext, die Höhe und Tiefe werden als Null betrachtet. 89
(siehe auch `\vphantom`, `\phantom` und `\smash`)
D `\def\hphantom{\v@false \h@true \ph@nt }`

∗ `\hrule` zieht eine waagerechte Linie. Dieser Befehl ist nur im vertikalen Modus anwendbar. In einem Absatz bewirkt er ein Absatzende. Syntax: *50*, 53, *82*, *136*
`\hrule width` *dimen* `height` *dimen* `depth` *dimen*
Die Parameter sind optional, sie geben Breite und Höhe und Tiefe des zu erzeugenden Strichs an. `\hrule` ohne Parameter erzeugt einen Strich mit der Breite der umgebenden Box und der Höhe 0.4 pt.

`\hrulefill` füllt eine Box mit einem waagerechten Strich auf. 106
Beispiel: `\hbox to 3cm{(\hrulefill )}` gibt
(———————)
(siehe auch `\dotfill`, `\leftarrowfill`, `\rightarrowfill`, `\downbracefill`, `\upbracefill`)
D `\def\hrulefill{\leaders \hrule \hfill }`

∗ `\hsize` Internes Register *horizontal size*. 21, 37, 38, *45*, *131*,
Dieses Register beinhaltet die Zeilenlänge, nach der der Zeilenumbruch erfolgt. Er kann lokal zum Beispiel in einer `\vbox` umgestellt werden.
Vorbesetzt: `\hsize=6.5 true in`
(siehe auch `\vsize`)

∗ `\hskip` erzeugt horizontalen Leerraum entsprechend der als Parameter angegebenen Länge, die auch dynamische Anteile enthalten darf. Durch zum Beispiel `\hskip 1cm` wird horizontaler Leerraum von 1 cm gesetzt (siehe auch `\vskip`). 29

∗ `\hss` erzeugt horizontalen dynamischen Leerraum, der beliebig wachsen und schrumpfen kann. Wird er in einer überfüllten Box gesetzt, so unterbleibt die Fehlermeldung. *48*, 142

∗ `\ht` referiert die Höhe eines Box-Registers (0 . . . 255) 134, *146*
Nach `\setbox0{\vrule height2cm width 0.4pt}` hat `\ht0` den Wert 2 cm.
Die Höhe einer Box kann durch Zuweisung auch extern wieder geändert werden
(siehe auch `\wd` für *width* und `\dp` für *depth*).

∗ `\hyphenation` trägt ein Wort in das Ausnahmelexikon ein. 52
Beispiel: `\hyphenation{tut-anch-amun}`
In dem einzutragenden Wort dürfen keine Konstruktionen mit Akzentbefehlen wie `\"` enthalten sein. Die Größe des Ausnahmelexikons ist auf ca. 300 Wörter beschränkt (implementationsabhängig).

∗ `\hyphenchar` ist die fontspezifische Nummer des Zeichens, das als Trennsymbol bei Trennungen verwendet wird. Normalerweise ist dies ein einfacher Trennstrich.
Nach `\hyphenchar\tenrm=-1` wird in der Schrift `\tenrm` getrennt, ohne daß eine Trennsymbol erscheint
(siehe auch `\defaulthyphenchar`).

∗ `\hyphenpenalty` Minuspunkte, die für eine einfache Trennung beim Absatzumbruch aufgerechnet werden. 53
Vorbesetzt: `\hyphenpenalty=50`

`\i` Textsatz: liefert (i ohne Punkt) — ı— 25
Mathematiksatz: `$\vec\imath$` für — $\vec{\imath}$ —
D `\chardef\i="10`

`\ialign` ist ein Hilfsmakro von plain-TEX, das einen `\halign`-Befehl mit `\tabskip=0pt` zu Beginn gibt.
D `\def\ialign{\everycr {}\tabskip \z@skip \halign }`

∗ `\if` prüft, ob die beiden folgenden *token* übereinstimmen. 115, *120*
Dabei werden evtl. folgende Makroaufrufe expandiert, bis zwei nicht weiter expandierbare *token* entstehen
(siehe auch `\ifx`).

∗ `\ifcase` *case*-Konstruktion

```
\ifcase Zahl oder Zahlregister
        Befehle für 0
  \or   Befehle für 1
  \or   Befehle für 2
    ...
  \else Befehle für sonstige Fälle
\fi
```

Beispielsweise liefert das Makro `\Monat`

```
\def\Monat{\ifcase \the\month ???
   \or Januar\or Februar\or M\"arz\or April%
   \or Mai\or Juni\or Juli\or August%
   \or September\or Oktober\or November%
   \or Dezember\else ???\fi}
```

den aktuellen Monat im Klartext.

∗ `\ifcat` prüft, ob die Kategorie-Codes (`\catcode`) der beiden folgenden *token* übereinstimmen. 115

∗ `\ifdim` prüft die Größenverhältnisse der folgenden Längenangaben: 115
`\ifdim` *dimension*$_1$ `<` *dimension*$_2$ `... \else ... \fi`
`\ifdim` *dimension*$_1$ `=` *dimension*$_2$ `... \else ... \fi`
`\ifdim` *dimension*$_1$ `>` *dimension*$_2$ `... \else ... \fi`

∗ `\ifeof` prüft, gefolgt von entweder Zahl 0 . . . 15, der Stromnummer einer Eingabedatei, oder der durch `\newread` benannten Nummer, ob noch weitere Information einzulesen ist. Es liefert *true*, falls bereits das Dateiende erreicht ist. 194

`\iff` Mathematiksatz: Relation — ⟺ — 87
D `\def\iff{\;\Longleftrightarrow \;}`

∗ `\iffalse` liefert eine *if*-Abfrage, die immer 'falsch' ist. Dieser Befehl wird gelegentlich in Makros verwendet, um ein überflüssiges `\fi` abzusättigen. 117
Er wird insbesondere automatisch bei `\newif` verwendet. Nach beispielsweise `\newif\ifABC` wird `\ABCfalse` durch `\def\ABCfalse{\let\ifabc=\iffalse}` belegt. Das Gegenstück dazu ist `\iftrue`.

∗ `\ifhbox` liefert, gefolgt von der Nummer eines Box-Register, *true*, falls diese Box eine `\hbox` enthält. 117

∗ `\ifhmode` *true*, falls man sich im *restricted horizontal mode*, das heißt im Innern einer `\hbox`, oder im *horizontal mode*, das heißt im Absatzumbruch, befindet. 116

∗ `\ifinner` testet auf *internal mode*: Dieser ist im *internal vertical mode*, also im Innern einer `\vbox`, oder im *restricted horizontal mode*, also im Innern einer `\hbox`, gesetzt. 116

∗ `\ifmmode` testet auf Mathematikmodus. 116, *116*

∗ `\ifnum` bildet einen Größenvergleich zwischen zwei Zahlenangaben: 115
`\ifnum` *zahl*$_1$ `<` *zahl*$_2$ `... \else ... \fi`
`\ifnum` *zahl*$_1$ `=` *zahl*$_2$ `... \else ... \fi`
`\ifnum` *zahl*$_1$ `>` *zahl*$_2$ `... \else ... \fi`
Beispiel: `\ifnum\pageno>10`
`\ifnum\count2<\count3`

∗ `\ifodd` prüft, ob die folgende Zahlenangabe *ungerade* ist. *48*, 115, *115*
Beispiel: `\ifodd\pageno`

*	`\iftrue`	liefert eine *if*-Abfrage, die immer 'wahr' ist. Dieser Befehl wird gelegentlich in Makros verwendet, um ein überüssiges `\fi` abzusättigen. Er wird insbesondere automatisch bei `\newif` verwendet. Nach beispielsweise `\newif\ifabc` wird `\abctrue` durch `\def\abctrue{\let\ifabc=\iftrue}` definiert. Das Gegenstück dazu ist `\iffalse` (siehe auch `\newif`).	117
*	`\ifvbox`	testet, gefolgt von der Nummer eines Box-Registers, ob in dem Box-Register eine `\vbox` enthalten ist. Diese Information ist wichtig, wenn eine Box mit `\unvbox` oder `\unhbox` wieder ausgegeben werden soll.	117
*	`\ifvmode`	testet auf *vertical mode* oder *internal vertical mode*.	116
*	`\ifvoid`	prüft, gefolgt von der Nummer eines Box-Registers, ob die Box leer ist (*true*, falls leer). *Leer* bedeutet dabei, daß das Box-Register weder eine `\hbox` noch eine `\vbox` enthält. Beispielsweise ein Boxregister mit einer leeren `\hbox` nach `\setbox0=\hbox{}` ist *nicht* leer. Das Box-Register 0 kann beispielsweise durch eine Ausgabe `{\setbox0=\hbox{\box0}}` in einer Klammergruppe wirklich geleert werden, ohne daß weitere Seiteneffekte auftreten.	117
*	`\ifx`	prüft *ohne volle Expandierung*, ob die beiden folgenden *token* die gleiche Bedeutung besitzen. Damit kann zum Beispiel getestet werden, ob die Definitionen zweier Makros gleich sind, oder ob ein Makroparameter leer ist, indem gegen ein leeres Makro getestet wird.	115, *116*, *118*, *120*
*	`\ignorespaces`	ist ein eingebauter Primitiv-Befehl, der alle in der Eingabe folgenden Leerzeichen überliest.	*45*
	`\Im`	Mathematiksatz: normales Zeichen — $\Im$ — D `\mathchardef\Im="23D`	88
	`\imath`	Mathematiksatz: normales Zeichen — $\imath$ — `\imath` liefert 'i' ohne Punkt. `$\vec\imath$` liefert — $\vec{\imath}$ — D `\mathchardef\imath="17B`	88
	`\immediate`	Dieser Befehl wirkt nur auf die direkt folgenden Befehle `\write`, `\openout`, `\closeout`. Normalerweise werden diese drei Befehle und ihre Parameter zwischengespeichert und erst während der effektiven Ausgabe einer Seite in der *Output-Routine* ausgeführt. Die Makros, die als Parameter mitgegeben sind, werden bei einem Aufruf ohne `\immediate` erst in der *Output-Routine* expandiert. Durch ein vorangestelltes `\immediate` wird die sofortige Expandierung und Ausführung bewirkt.	196

∗ `in` Längeneinheit Zoll (inch) 22
1 in = 2,54 cm = 72,27 pt

`\in` Mathematiksatz: liefert — ∈ — 86
(Gegenstück `\ni` — ∋ —)
D `\mathchardef\in="3232`

∗ `\indent` Mit diesem Befehl wird normalerweise ein neuer Absatz begonnen, nach einer Leerzeile oder nach `\par`. Es wird dann ein Einzug vom Umfang des Parameters `\parindent` gesetzt. 28
Zwei `\indent`-Befehle hintereinander verdoppeln den Einzug
(siehe auch `\noindent`, das einen Absatz ohne Einzug beginnt).

`\inf` Mathematiksatz: großer Operator — inf — 78
D `\def\inf{\mathop {\rm inf}}`

`\infty` Mathematiksatz: normales Zeichen — ∞ — 88
D `\mathchardef\infty="231`

∗ `\input` wechselt die Befehlseingabe auf eine andere Datei. Nach `\input` folgt der Dateiname. 191, 193
Beispiel: `\input myfile`

∗ `\insert` Damit kann vertikales Material in ein *insertion register* eingefügt werden. Mit diesem Befehl arbeiten das Fußnotenmakro und Anweisungen wie `\topinsert`. 49
Syntax: `\insert` *n* `{` *vertikales Material* `}`
Die Einfügungsregister sind mit allen anderen Registern mit der gleichen Nummer gekoppelt. Auf jeden Fall sollte man sich zum Beispiel mit `\newinsert\myinsert` die Nummer eines freien *insertion register* besorgen.

∗ `\insertpenalties` 172
enthält während der *output-routine* die Anzahl noch nicht abgearbeiteter Einfügungen. Während der eigentlichen `\insert`-Operation werden in diesem Register die internen Gewichte für die Einfügung gehalten.

`\int` Mathematiksatz: großer Operator — ∫ — 73
(siehe auch `\smallint`, `\intop`)
D `\def\int{\intop \nolimits }`

`\interdisplaylinepenalty`
Register der plain-TEX-Makros: Minuspunkte, die für das Umbrechen einer mit `\displaylines` entstandenen Formelfolge über eine Seitengrenze hinweg berechnet werden.
D
```
\newcount\interdisplaylinepenalty
\interdisplaylinepenalty=100
```

`\interfootnotelinepenalty`
Plain-TEX-Register für die Minuspunkte, die für das Umbrechen einer Fußnote über Seitengrenzen hinweg berechnet werden. `\footnote` setzt die `\interlinepenalty` auf diesen Wert.
D
```
\newcount\interfootnotelinepenalty
\interfootnotelinepenalty=100
```

∗ `\interlinepenalty`
Minuspunkte für den Umbruch eines Absatzes über eine Seitengrenze.
Vorbesetzt: `\interlinepenalty=0`
(Das Fußnotenmakro setzt diesen lokal auf `\interfootnotelinepenalty`)

`\intop` Mathematiksatz: großer Operator — $\int$ — 73
Im Gegensatz zu `\int` werden untere und obere Grenzen direkt unter und über das Symbol gesetzt (siehe auch `\int`).
D
```
\mathchardef\intop="1352
```

`\iota` Mathematiksatz: griechischer Buchstabe — ι — 66
D
```
\mathchardef\iota="113
```

`\it` Einstellen der Schrift(-familie) *italic*. 56
D
```
\def\it{\fam \itfam \tenit }
```

`\item` erzeugt eine Aufzählungsliste mit `\parindent` als Einzug. 42ff
D
```
\def\item{\par \hang \textindent }
```

`\itemitem` erzeugt eine Aufzählungsliste mit 2 × `\parindent` als Einzug. 42ff
D
```
\def\itemitem{\par \indent \hangindent 2\parindent
        \textindent }
```

`\itfam` ist die interne Nummer (4) der *italic*-Schriftfamilie. (siehe auch `\newfam`, `\it`)
D
```
\newfam\itfam
\def\it{\fam\itfam\tenit}
```

`\j` Textsatz: liefert — ȷ — 25
D
```
\chardef\j="11
```

`\jmath` Mathematiksatz: liefert — $\jmath$ — 88
Beispiel: `$\vec\jmath$` liefert — $\vec{\jmath}$ —
D
```
\mathchardef\jmath="17C
```

∗ `\jobname` ist der Name des laufenden TEX-Auftrages. So wird aus dem Inhalt von `\jobname` und den Namenserweiterungen "`log`" und "`dvi`" der Name der Ausgabedateien bestimmt. Allerdings sind hier häufig Besonderheiten festzustellen, die durch das Betriebssystem und der jeweiligen Implementierung verursacht werden. 192

`\joinrel` Hilfsmakro in plain-TEX zur Bildung der Befehle für 'lange' Pfeile, wie etwa `\longrightrightarrow`. Diese werden aus mehreren Stückchen zusammengesetzt.
D `\def\joinrel{\mathrel {\mkern -3mu}}`

`\jot` ist ein plain-TEX-Längenregister, belegt mit 3 pt.
D `\newdimen\jot  \jot=3pt`

`\kappa` Mathematiksatz: griechischer Buchstabe — κ — 66
D `\mathchardef\kappa="114`

`\ker` Mathematiksatz: großer Operator — ker — 78
D `\def\ker{\mathop {\rm ker}\nolimits }`

∗ `\kern` liefert, gefolgt von einer Längenangabe, je nach aktuellem Modus vertikalen oder horizontalen *kern*, das heißt Leerraum, an dem weder ein Zeilen- noch ein Seitenwechsel erfolgen kann. *50*
(Für den Mathematiksatz muß `\mkern` verwendet werden.)

`\l` Textsatz: (polnisches ł) liefert — ł — 25
D `\def\l{\char 321}` % für computer modern

`\L` Textsatz: (polnisches Ł) liefert — Ł — 25
In 256-Zeichenfonts ist dies ein eigenständiges Zeichen.
D
```
\def\L{\leavevmode \setbox 0\hbox {L}%
       \hbox to\wd 0{\hss \char 32L}}
```

`\lambda` Mathematiksatz: griechischer Buchstabe — λ — 66
D `\mathchardef\lambda="115`

`\Lambda` Mathematiksatz: griechischer Buchstabe — Λ — 67
D `\mathchardef\Lambda="7003`

`\land` Mathematiksatz: binärer Operator — $\wedge$ — 86
(äquivalent ist `\wedge`)
D `\mathchardef\land="225E`

`\langle` Mathematiksatz: öffnende Klammer — $\langle$ — 75
(Gegenstück: `\rangle`)
D `\def\langle{\delimiter "426830A }`

3 ∗ `\language` steuert nach welcher Trenntabelle aktuell getrennt werden soll. Es sind maximal 256 Trenntabellen (theoretisch) gleichzeitig möglich. Durch eine Zuweisung `\language=` n mit $n = 0 \ldots 255$ wird die aktuelle Trenntabelle umgestellt. Welche Trenntabellen gleichzeitig vorhanden sind, hängt von dem verwendeten Formatfile ab. Wird ein Wert verwendet, zu dem keine Trennmuster geladen sind, wird gar nicht getrennt. Es ist sogar möglich, innerhalb des gleichen Absatzes mehrfach die Sprache zu wechseln. 51

* `\lastbox` liefert im *internal vertical mode* und in beiden *horizontal modes* die letzte `\vbox` oder `\hbox` zurück, falls direkt vor diesem Befehl eine solche Box gebildet wurde. Durch Zuweisung, wie etwa `\setbox0=\lastbox`, enthält das Box-Register 0 diese Box. Sie kann dann auch weiterverarbeitet werden. Durch die `\lastbox`-Operation wird die Box entfernt.

* `\lastkern` liefert, falls das letzte Element eine `\kern`-Operation war, dieses aus. Das Element bleibt aber erhalten. Durch `\kern-\lastkern` kann es aber rückgängig gemacht werden. Dann sind allerdings zwei `\kern`-Elemente hintereinander vorhanden.
Durch `\unkern` wird der `\kern` explizit entfernt.

* `\lastpenalty` liefert, falls das letzte Element ein `\penalty`-Eintrag ist, diesen Wert zurück.
Durch `\unpenalty` kann das `\penalty`-Element entfernt werden. Durch etwa `\count7=\lastpenalty` kann das `\penalty`-Element zur Inspektion auf ein Register zugewiesen werden.

* `\lastskip` liefert, falls das letzte Element ein *skip*-Eintrag ist, dessen Wert zurück.
Durch `\unskip` kann dieses Element wieder entfernt werden.
Durch etwa `\skip6=\lastskip` wird das *skip*-Element zur Inspektion einem Register zugewiesen.

`\lbrace` Mathematiksatz: öffnende Klammer — { — 75
wachsend in Kombination mit `\big..`, `\left`, `\right`.
Der Befehl `\{` ist dazu äquivalent.
(Das Gegenstück ist `\rbrace` bzw. `\}`)
D `\def\lbrace{\delimiter "4266308 }`

`\lbrack` Mathematiksatz: öffnende Klammer — [— 75
wachsend in Kombination mit `\big..`, `\left`, `\right`.
Der Befehl `[` ist dazu äquivalent (siehe auch `\rbrack`).
D `\def\lbrack{[}`

* `\lccode` *(lowercase code)* Zeichen, das jedem der möglichen 256 Zeichen zugeordnet ist. Es bestimmt, in welches Symbol das betreffende Zeichen beim `\lowercase`-Befehl umgewandelt wird. So ist `\lccode‘\A=‘\a` definiert (siehe auch `\uccode` für `\uppercase`).

`\lceil` Mathematiksatz: öffnende Klammer — ⌈ — 75
(siehe auch `\rceil` und `\lfloor`, `\rfloor`)
D `\def\lceil{\delimiter "4264306 }`

`\ldotp` Mathematiksatz: Punkt als Satzzeichen (wird in `\ldots` weiterverwendet)
D `\mathchardef\ldotp="613A`

`\ldots` Mathematiksatz: *(lower dots)* liefert — … — (siehe auch `\vdots`, `\cdots`, `\ddots`, `\dots`) 80
D `\def\ldots{\mathinner {\ldotp \ldotp \ldotp }}`

`\le` Mathematiksatz: Relation — ≤ — 86
D `\mathchardef\le="3214`

∗ `\leaders` wiederholt die folgende Box oder `\hrule` bis zur Breite, die durch das folgende `\hskip`, bzw. `\hfill` angegeben wird. Beispiel:

```
\def\leaderfill{%
  \leaders\hbox to 1em{\hss.}\hfill}
```

`\leaderfill` liefert angewendet:
Das ist . . . . . .
nicht der Anfang vom Ende . . . . . .
Dabei werden die entstehenden Boxen so ausgerichtet, daß sie in aufeinanderfolgenden Zeilen auch untereinander gleich ausgerichtet sind
(siehe auch `\cleaders` *centered*, `\xleaders` *expanded*).

`\leavevmode` beginnt einen Absatz, falls noch keiner angefangen war. Der Vorteil ist, daß zwar ein neuer Absatz beginnt, aber noch keine Absatzeinrückung gesetzt wird. Eine folgende Box wird dann ohne eine Einrückung plaziert, ein `\indent` Befehl erzeugt den üblichen Einzug. 185
D `\def\leavevmode{\unhbox \voidb@x }`

∗ `\left` Mathematiksatz: `\left` und `\right`-Befehle klammern im Mathematiksatz eine Unterformel ein. Dabei folgen auf `\left` und `\right` jeweils noch Angaben für die Begrenzer, mit denen die Unterformel eingeklammert werden soll. Diese Begrenzer werden dann so groß wie die eingeschlossene Unterformel. 77, *79*, *84*
Beispiel:
`$$ \left( x \over x+1 \right) $$` liefert

$$\left(\frac{x}{x+1}\right)$$

`\leftarrow` Mathematiksatz: Relation liefert — ← — (`\gets` ist äquivalent.) 87
D `\mathchardef\leftarrow="3220`

`\Leftarrow` Mathematiksatz: Relation liefert — ⇐ — 87
D `\mathchardef\Leftarrow="3228`

`\leftarrowfill` Textsatz: erzeugt einen Pfeil mit der Länge, wie es die umgebende Box fordert. 106
Beispiel:

```
\hbox to 4cm{X \leftarrowfill\ Y}
```
X ←——————— Y

(siehe auch `\rightarrowfill`,`\hrulefill`)

D
```
\def\leftarrowfill{$\m@th \mathord \leftarrow
    \mkern -6mu \cleaders
      \hbox {$\mkern -2mu\mathord -\mkern-2mu$}\hfill
    \mkern -6mu\mathord -$}
```

`\leftharpoondown` 87
Mathematiksatz: Relation — ↽ —
D `\mathchardef\leftharpoondown="3129`

`\leftharpoonup` Mathematiksatz: Relation — ↼ — 87
D `\mathchardef\leftharpoonup="3128`

3 ∗ `\lefthyphenmin` ist ein neues internes Register, das angibt, wie viele Zeichen beim Trennen mindestens in der vorangehenden Zeile noch übrig bleiben müssen. Das entsprechende Gegenstück ist `\righthyphenmin`.
D `\lefthyphenmin=2 \righthyphenmin=3`

`\leftline` Textsatz: Der Inhalt von `\leftline{..text..}` wird linksbündig als Zeile gesetzt. Dieser Befehl sollte nur im vertikalen Modus, also außerhalb von Absätzen, verwendet werden. *18*, *29*
(siehe auch `\rightline`, `\centerline`)
D
```
\def\leftline#1{\line {#1\hss }}
\def\line{\hbox to\hsize}
```

`\leftrightarrow` 87
Mathematiksatz: Relation — ↔ —
D `\mathchardef\leftrightarrow="3224`

`\Leftrightarrow` 87
Mathematiksatz: Relation — ⇔ —
D `\mathchardef\Leftrightarrow="322C`

∗ `\leftskip` Textsatz: Durch Zuweisung einer Längenangabe an die Variable `\leftskip` werden alle Absatzzeilen mit einer gleichbleibenden linksseitigen Einrückung versehen. *33*, 36ff, 38
Voreingestellt: `\leftskip=0pt`
(siehe auch `\rightskip`, `\narrower`)

`\leq` Mathematiksatz: Relation — ≤ — 86
(äquivalent zu `\le`)
D `\mathchardef\leq="3214`

`\leqalignno` Mathematiksatz: Makro zum Ausrichten von Formeln mit linksbündiger Numerierung 85

```
\leqalign{ 1. Teil & 2. Teil  & nr \cr
                          ...
           1. Teil & 2. Teil  & nr \cr}
```

sorgt dafür, daß die zweiten Teile bündig untereinander stehen.
Beispiel:

```
$$\leqalignno{a&=b+c&(1)\cr a-b&=c&(2)\cr}$$
```

$$a = b + c \tag{1}$$
$$a - b = c \tag{2}$$

(siehe auch `\eqalignno`, `\eqalign`)

D
```
\def\leqalignno#1{\displ@y \tabskip\centering
    \halign to\displaywidth{\hfil$\@lign
          \displaystyle{##}$\tabskip\z@skip
       &$\@lign\displaystyle{{}##}$\hfil
                 \tabskip\centering
       &\kern-\displaywidth
         \rlap{$\@lign##$}\tabskip\displaywidth\crcr
    #1\crcr}}
```

∗ `\leqno` Mathematiksatz: 84
In einer *display-style*-Formel verwendet, wird damit eine linksbündige Numerierung erzeugt.
`$$` *Formel* `\leqno` *Numerierung* `$$`
(siehe auch `\eqno`)

∗ `\let` definiert eine neue Kontrollsequenz durch *Kopieren* der alten Bedeutung. Durch 112, *120*, *198*

```
\let\bs=\bigskip
\def\bigskip{\vskip 24pt}
```

bleibt unter dem Befehl `\bs` der alte `\bigskip`-Befehl erhalten, `\bigskip` dagegen ist neu definiert.

`\lfloor` Mathematiksatz: öffnende Klammer — ⌊ — 75
(siehe auch `\rfloor`, `\lceil` und `\rceil`)
D `\def\lfloor{\delimiter "4262304 }`

`\lg` Mathematiksatz: großer Operator — lg — 78
D `\def\lg{\mathop {\rm lg}\nolimits }`

`\lgroup` Mathematiksatz: öffnende Klammer — (— 75, *77*
(nur mit `\big..`, `\left`, `\right` verwendbar)
(Gegenstück: `\rgroup`)
D `\def\lgroup{\delimiter "400033A }`

`\lhook` Mathematiksatz: Hilfszeichen bei der Konstruktion von `\hookrightarrow`. `\lhook` liefert — ↪ —
(siehe auch `\rhook`).
D `\mathchardef\lhook="312C`

`\lim` Mathematiksatz: großer Operator — lim — 78
D `\def\lim{\mathop {\rm lim}}`

`\liminf` Mathematiksatz: großer Operator — lim inf — 78
D `\def\liminf{\mathop {\rm lim\,inf}}`

∗ `\limits` Mathematiksatz: verändert das Satzverhalten für Indizes und Exponenten, falls es einem mathematischen Operator direkt nachgestellt wird. Exponent und Index werden dann *direkt* über und unter das Symbol gesetzt. 74
Beispiel:
`$$\int\limits_0^\pi \sin x$$`

$$\int\limits_0^\pi \sin x \qquad \text{normal:} \quad \int_0^\pi \sin x$$

(siehe auch `\nolimits`, `\displaylimits`)

`\limsup` Mathematiksatz: großer Operator — lim sup — 78
D `\def\limsup{\mathop {\rm lim\,sup}}`

`\line` Textsatz: Der Befehl `\line{..text..}` erzeugt im Textsatz eine einzelne Ausgabezeile. Durch geeigneten dynamischen Leerraum sollte die gebildete Box aufgefüllt werden.
D `\def\line{\hbox to\hsize }`

∗ `\linepenalty` Minuspunkte, die für jede Zeile als 'Grundlast' berechnet werden. Wird dieser Wert erhöht, so versucht das Programm, die Zeilenzahl für einen Absatz möglichst klein zu halten.
Vorbesetzt: `\linepenalty=10`

∗ `\lineskip` vertikaler Mindestabstand zwischen der Unterkante einer Box und der Oberkante der folgenden Box, der gesetzt wird, falls `\lineskiplimit` unterschritten wird. 30, 145
Vorbesetzt: `\lineskip=1pt`

∗ `\lineskiplimit` Wert für den vertikalen Abstand, den zwei aufeinanderfolgende Boxen zwischen Unterkante einer Box und der Oberkante der folgenden mindestens haben müssen. Wird dieser unterschritten, wird als Abstand der Wert von `\lineskip` verwendet. 30, 145
Vorbesetzt: `\lineskiplimit=0pt`

`\ll` Mathematiksatz: Relation — ≪ — 86
(siehe auch `\gg` für — ≫ —)
D `\mathchardef\ll="321C`

`\llap` *left lap* 142
Ausgabe von nach links überlappender Information.
Beispiel: `ooo\llap{//}uuu` liefert — oo/uuu —
(siehe auch `\rlap` für: ooo/u)
D `\def\llap#1{\hbox to\z@ {\hss #1}}`

`\lmoustache` Mathematiksatz: Klammer — $\left\lmoustache\right.$ — 75
Sie kann nur mit `\big..`, `\left` und `\right` verwendet werden
(siehe auch `\rmoustache`).
D `\def\lmoustache{\delimiter "4000340 }`

`\ln` Mathematiksatz: großer Operator — ln — 78
D `\def\ln{\mathop {\rm ln}\nolimits }`

`\lnot` Mathematiksatz: normales Zeichen — ¬ —
(äquivalent zu `\neg`)
D `\mathchardef\lnot="23A`

`\log` Mathematiksatz: großer Operator — log — 78
D `\def\log{\mathop {\rm log}\nolimits }`

* `\long` Befehl, der einer `\def`, `\gdef` oder `\edef` Anweisung vorangestellt wird: Er erlaubt die Benutzung von ganzen Absätzen als Parameter. 114

`\longleftarrow` Mathematiksatz: Relation — ⟵ — 87
D `\def\longleftarrow{\leftarrow\joinrel\relbar}`

`\Longleftarrow` Mathematiksatz: Relation — ⟸ — 87
D `\def\Longleftarrow{\Leftarrow\joinrel\Relbar}`

`\longleftrightarrow` 87
Mathematiksatz: Relation — ⟷ —
D
```
\def\longleftrightarrow
        {\leftarrow\joinrel\rightarrow}
```

`\Longleftrightarrow` 87
Mathematiksatz: Relation — ⟺ —
D
```
\def\Longleftrightarrow
          {\Leftarrow\joinrel\Rightarrow}
```

`\longmapsto` Mathematiksatz: Relation — ⟼ — 87
D `\def\longmapsto{\mapstochar \longrightarrow }`

`\longrightarrow` 87
Mathematiksatz: Relation — ⟶ —
D `\def\longrightarrow{\relbar\joinrel\rightarrow}`

`\Longrightarrow` 87
Mathematiksatz: Relation — ⟹ —
D `\def\Longrightarrow{\Relbar\joinrel\Rightarrow}`

`\loop` plain-TEX-Makro zur Schleifenkonstruktion. 126ff, *140*
Syntax: `\loop` α `\if..` β `\repeat`
α, β sind beliebige Kommandofolgen, sie dürfen aber auch leer sein. `\if..` ist irgendeine der *if*-Abfragen, `\repeat` ist das Ende der Schleife. Zu Beginn wird α ausgeführt. Ist die *if*-Anweisung *true*, wird anschließend β ausgeführt, ist es dagegen *false*, wird abgebrochen. Nach `\repeat` beginnt der Prozeß wieder bei α.
Die Teile α und β dürfen selbst keine `\if`-Anweisungen enthalten, können aber Makros aufrufen, die wieder `\if`-Befehle verwenden.

```
\def\loop#1\repeat {\def \body {#1}\iterate }
\def\iterate{\body \let\next\iterate
             \else\let\next\relax\fi \next}
\let\repeat=\fi
```

∗ `\looseness` Optimierungsparameter für den Absatzumbruch.
Dieser ist mit Null voreingestellt; wird er auf eins oder zwei erhöht, so versucht das TEX-Programm, den Absatzumbruch so zu gestalten, daß die Absätze um ein oder zwei Zeilen länger werden. `\looseness` wird durch TEX automatisch zu Beginn eines Absatzes auf Null zurückgesetzt.

`\lor` Mathematiksatz: binärer Operator — ∨ — 86
(äquivalent ist `\vee`)

```
\mathchardef\lor="225F
```

∗ `\lower` Damit kann eine Box im *horizontal mode* gesenkt werden. *104*, 143, *155*
Vorher stand der Befehl `\lower3pt\hbox{Box}`
(siehe auch `\raise`, `\moveleft`, `\moveright`).

∗ `\lowercase` bildet Kleinbuchstaben aus der Eingabe.
Durch `\lowercase{aBc}` wird die Eingabe zu `abc`. Makros, die in der Eingabe enthalten sind, werden *nicht* expandiert. Dies wird durch folgende Konstruktion erreicht:

```
\edef\next{...information...}
\lowercase\expandafter{\next}
```

(siehe auch `\uppercase`)

`\lq` Ersatzbefehl (*left quote*)für — ‘ — (Akzentzeichen) 16

```
\def\lq{‘}
```

∗ `\mag` TEX internes Register für die globale Vergrößerung, dieses wird mit dem `\magnification`-Befehl belegt.

`\magnification` setzt die globale Vergrößerung des Dokuments. Im Normalfall werden die durch `\magstep` angebotenen Vergrößerungsfaktoren verwendet. (Ohne Berücksichtigung des Satzspiegels `\hsize` und `\vsize`) 27, 57

```
D \def\magnification{\afterassignment \m@g \count@ }
  \def\m@g{\mag\count@
    \hsize6.5truein \vsize8.9truein
    \dimen\footins8truein}
```

`\magstep` bildet mit den Parameter $0 \ldots 5$ die Vergrößerungsfaktoren $(1000 * 1, 2^n)$, also zum Beispiel `\magstep2`. 23, 57

```
D \def\magstep#1{\ifcase #1 \@m \or 1200\or 1440\or
      1728\or 2074\or 2488\fi \relax }
```

`\magstephalf` Vergrößerungsfaktor von '1095' ($\approx \sqrt{1,2}$) 57

```
D \def\magstephalf{1095 }
```

`\makefootline` Hilfsmakro der Standard-Output-Routine zur Bildung der Seitenfußzeile. (Diese enthält in der Voreinstellung die Seitennummer zentriert in der Zeile.) 174

```
D \def\makefootline{\baselineskip 24\p@
      \line {\the \footline}}
```

`\makeheadline` Hilfsmakro der Standard-Output-Routine zur Bildung der Seitenkopfzeile. (Diese ist in der Voreinstellung leer.) 173, *175*

```
D \def\makeheadline{\vbox to\z@{\vskip -22.5\p@
      \line{\vbox to8.5\p@{}%
            \the\headline}\vss}\nointerlineskip}
```

`\mapsto` Mathematiksatz: Relation — ↦ — 87

```
D \def\mapsto{\mapstochar \rightarrow }
```

`\mapstochar` Hilfszeichen zur Bildung von `\mapsto`, es enthält — ˫—

```
D \mathchardef\mapstochar="3237
```

∗ `\mark` Damit wird ein Text in ein globales Merkregister eingetragen. Dieser Text — interessant ist es nur, wenn dies mehrfach hintereinander, etwa mit Kapitelüberschriften, geschieht — kann nun mit Bezug auf die aktuelle Seite abgefragt werden: `\botmark` liefert den letzten Eintrag auf der Seite, `\topmark` den letzten Eintrag der Vorgängerseite und `\firstmark` den ersten auf der aktuellen Seite. Die Anwendung ist eigentlich nur während der *output-routine*, etwa in den Befehlen `\headline` oder `\footline`, sinnvoll (siehe auch `\botmark`, `\firstmark`, `\topmark`). 177ff

∗ `\mathaccent` TEX-Primitivbefehl zur Bildung mathematischer Akzente. So ist `\ddot` definiert durch

```
\def\ddot{\mathaccent"707F }
```

Die Syntax der nachfolgenden Zahlangabe ist unter dem Befehl `\mathchar` zu finden.

∗ `\mathbin` Mathematiksatz: erzwingt das Setzen des folgende Zeichens mit dem satztechnischen Verhalten eines *binären Operators*. So wird durch `\mathbin=` das Gleichheitszeichen "=" wie ein binärer Operator behandelt. 88, 151ff

∗ `\mathchar` Mathematiksatz: Damit wird ein mathematisches Zeichen direkt aus einer Codetabelle angesteuert und mit einer bestimmten satztechnischen Funktion versehen. `\mathchar"1350` gibt das Zeichen `"50` (hexadezimal), dies ist das Summenzeichen, aus. Dabei wird es als (großer) Operator (Klasse `1`) aus der Schriftfamilie `3` *(math extension fonts)* genommen.
Die Syntax ist `\mathchar"`*c f hh*, mit *c* als Klassenangabe, *f* als Schriftfamilienangabe und *hh* als Position in der Codetabelle. (Die Angaben sind hexadezimal.)
Die Klassenangaben ergeben sich aus:

0 normale Zeichen (α)
1 (große) Operatoren ($\sum$)
2 binäre Operatoren ($+$)
3 Relationen ($<$)
4 öffnende Klammern ({)
5 schließende Klammern (})
6 mathematisch Satzzeichen (.)
7 variable Klassennummer:
Diese entspricht Null, wenn nicht ein Befehl zum Wechsel der Schriftfamilie, etwa `\bf`, auftritt. Diese ersetzt dann automatisch die Klasse 7
(siehe auch `\fam`, `\delimiter`).

∗ `\mathchardef` Mathematiksatz: Durch z. B. `\mathchardef\sum="1350` wird ein mathematisches Zeichen benannt und seine Funktion festgelegt. Die Syntax für die Zahlenangabe entspricht `\mathchar`.

∗ `\mathchoice` Mathematiksatz: Dieser Befehl besitzt 4 Parameter: 155
`\mathchoice{`*display*`}{`*text*`}{`*script*`}{`*scriptscript*`}`
Damit kann eine mathematische Formel für alle 4 Varianten des Mathematiksatzes vorbereitet werden. Das TEX-Programm sucht sich dann die Variante aus, die dem augenblicklich gültigen Arbeitsmodus entspricht. Das Ergebnis ist vom satztechnischen Typ einer Unterformel.

∗ `\mathclose` Mathematiksatz: erzwingt das Setzen des folgenden Zeichens mit dem satztechnischen Verhalten einer *schliessenden Klammer*. Durch `\mathclose:` wird beispielsweise aus ":" eine schließende Klammer. 88, 151ff

* `\mathcode` Mathematiksatz: weist einem Zeichen einen Code zu, der seine satztechnische Behandlung bestimmt.
Beispiel: `\mathcode‘\<="313C`
bedeutet, daß das Zeichen “<” als Relation gesetzt wird. Gefunden wird es in Schriftfamilie `1` an der Position `"3C`. Die führende “3” bestimmt das Satzverhalten als Relation. Gleichzeitig werden damit die einzelnen *Eingabezeichen* im Mathematiksatz auf andere Plätze in der jeweiligen Codetabelle umkodiert (siehe auch `\delimiter`).
INITEX setzt `\mathcode`$x = x$, für x =0..255 mit den Ausnahmen der Buchstaben: `\mathcode x=x+"7100` und der Ziffern `\mathcode x=x+"7000`. Dies bedeutet: Die Buchstaben werden im Mathematiksatz aus der Schriftfamilie (1), das heißt “*math italic*”, die Ziffern aus die Schriftfamilie (0), also “roman” genommen.

Folgende ASCII-Codes kleiner 32, das heißt Eingaben mit *control*, werden definiert:

`\mathcode‘\^^?="1273`	$\int$	`\smallint`
`\mathcode‘\^^@="2201`	$\cdot$	`\cdot`
`\mathcode‘\^^A="3223`	$\downarrow$	`\downarrow`
`\mathcode‘\^^B="010B`	α	`\alpha`
`\mathcode‘\^^C="010C`	β	`\beta`
`\mathcode‘\^^D="225E`	$\wedge$	`\land`
`\mathcode‘\^^E="023A`	$\neg$	`\lnot`
`\mathcode‘\^^F="3232`	$\in$	`\in`
`\mathcode‘\^^G="0119`	π	`\pi`
`\mathcode‘\^^H="0115`	λ	`\lambda`
`\mathcode‘\^^I="010D`	γ	`\gamma`
`\mathcode‘\^^J="010E`	δ	`\delta`
`\mathcode‘\^^K="3222`	$\uparrow$	`\uparrow`
`\mathcode‘\^^L="2206`	$\pm$	`\pm`
`\mathcode‘\^^M="2208`	$\oplus$	`\oplus`
`\mathcode‘\^^N="0231`	∞	`\infty`
`\mathcode‘\^^O="0140`	∂	`\partial`
`\mathcode‘\^^P="321A`	$\subset$	`\subset`
`\mathcode‘\^^Q="321B`	$\supset$	`\supset`
`\mathcode‘\^^R="225C`	$\cap$	`\cap`
`\mathcode‘\^^S="225B`	$\cup$	`\cup`
`\mathcode‘\^^T="0238`	$\forall$	`\forall`
`\mathcode‘\^^U="0239`	$\exists$	`\exists`
`\mathcode‘\^^V="220A`	$\otimes$	`\otimes`
`\mathcode‘\^^W="3224`	$\leftrightarrow$	`\leftrightarrow`
`\mathcode‘\^^X="3220`	$\leftarrow$	`\leftarrow`
`\mathcode‘\^^Y="3221`	$\rightarrow$	`\rightarrow`
`\mathcode‘\^^Z="8000`	$\neq$	`\ne`

```
\mathcode‘\^^[="2205   ◇  \diamond
\mathcode‘\^^\="3214   ≤  \le
\mathcode‘\^^]="3215   ≥  \ge
\mathcode‘\^^^="3211   ≡  \equiv
\mathcode‘\^^_="225F   ∨  \lor
```

Alle weiteren normalen Zeichen sind wie folgt belegt, wobei die verschiedenen Symbole aus unterschiedlichen Schriftfamilien genommen werden:

```
\mathcode‘\ ="8000     \mathcode‘\;="603B
\mathcode‘\!="5021     \mathcode‘\<="313C
\mathcode‘\’="8000     \mathcode‘\=="303D
\mathcode‘\(="4028     \mathcode‘\>="313E
\mathcode‘\)="5029     \mathcode‘\?="503F
\mathcode‘\*="2203     \mathcode‘\[="405B
\mathcode‘\+="202B     \mathcode‘\\="026E
\mathcode‘\,="613B     \mathcode‘\]="505D
\mathcode‘\-="2200     \mathcode‘\_="8000
\mathcode‘\.="013A     \mathcode‘\{="4266
\mathcode‘\/="013D     \mathcode‘\|="026A
\mathcode‘\:="303A     \mathcode‘\}="5267
```

`\mathhexbox` Mathematiksatz: internes Hilfsmakro zur Definition von `\dag`, `\S`, `\P`. Diese Befehle geben mathematische Symbole im normalen Textsatz aus.

```
\def\mathhexbox#1#2#3{\leavevmode
        \hbox {$\m@th \mathchar"#1#2#3$}}
```

∗ `\mathinner` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion einer *Unterformel*. 88, 151ff

∗ `\mathop` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion eines *großer Operator*. *78*, 88, 151ff

∗ `\mathopen` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion einer *öffnenden Klammer*. 88, 151ff

∗ `\mathord` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion eines *normalen Zeichens*. 88, 151ff

`\mathpalette` Mathematiksatz:
Syntax: `\mathpalette{`α`}{`β`}`
Dabei wird ein `\mathchoice`-Befehl erzeugt, wobei der α-Anteil unverändert, der β-Anteil abhängig vom Satzmodus `\displaystyle`, `\textstyle` ... gesetzt wird.
(Hilfsmakro zum Beispiel für den `\root`-Befehl.)

```
D \def\mathpalette#1#2{\mathchoice
        {#1\displaystyle {#2}}%
        {#1\textstyle {#2}}%
        {#1\scriptstyle {#2}}%
        {#1\scriptscriptstyle {#2}}}
```

* `\mathpunct` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion eines *Satzzeichens*. 88, 151ff

* `\mathrel` Mathematiksatz: erzwingt das Setzen der folgenden Information mit der satztechnischen Funktion einer *Relation*. 88, 151ff

`\mathstrut` Mathematiksatz: bildet eine leere Box mit der Höhe und Tiefe einer runden Klammer. Diese wird verwendet, um einen Mindestzeilenabstand zum Beispiel in Matrizen zu erreichen.

```
D \def\mathstrut{\vphantom(}
```

* `\mathsurround` Mathematiksatz: Variable, die angibt, wieviel zusätzlicher Leerplatz vor und nach einer Formel im *textstyle* (`$..$`) gesetzt werden soll. 151
Vorbesetzt: `\mathsurround=0pt`

`\matrix` Mathematiksatz: setzt eine Matrix ohne Klammern. 79ff
Beispiel:

```
$$\matrix{ 1 & 2 & 3 \cr
           4 & 5 & 6 \cr
           7 & 8 & 9 \cr}$$
```

liefert

$$\matrix{ 1 & 2 & 3 \cr 4 & 5 & 6 \cr 7 & 8 & 9 \cr}$$

(beachte: Intern wird mit `\halign` gesetzt!)

```
D \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}}}\,}
```

`\max` Mathematiksatz: großer Operator — max — 78

```
D \def\max{\mathop {\rm max}}
```

* `\maxdeadcycles` interner Parameter: setzt die Grenze, wie oft die Output-Routine hintereinander aufgerufen werden darf, ohne daß eine Ausgabe mittels `\shipout` erfolgt. 172
Vorbesetzt: `\maxdeadcycles=25`

∗ `\maxdepth` internes Register: maximale Unterlänge einer Box. 173
Vorbesetzt: `\maxdepth=4pt`

`\maxdimen` interner Parameter der plain-TEX Makros: Er enthält die maximal mögliche Längenangabe.
D
```
\newdimen\maxdimen
\maxdimen=16383.99999pt
```

∗ `\meaning` Protokollbefehl: gibt die Bedeutung eines Zeichens oder Befehls aus. 118, 199
`\meaning A` → 'the letter A'
`\meaning\par` → '`\par` ' *eingebauter Befehl*
`\meaning\bigskip`
→ '`macro:->\vskip \bigskipamount`'
Alle Zeichen der Ausgabe haben den `\catcode` 12, das heißt *sonstige Zeichen.*
In der Kombination mit `\message{\meaning ...}` erfolgt die Ausgabe auf dem Terminal und ins Protokoll (siehe auch `\show` und `\showthe`).

`\medbreak` Textsatz: gibt wie `\medskip` einen mittleren vertikalen Leerraum aus und setzt einen 'mittelguten' Umbruchpunkt für den Seitenumbruch 46
(siehe auch `\smallbreak`, `\bigbreak`).
D
```
\def\medbreak{\par \ifdim \lastskip <\medskipamount
      \removelastskip \penalty -100\medskip \fi }
```

∗ `\medmuskip` Mathematiksatz: Länge eines mittleren Leerraums. 78, 151
Diese (setzbare) Länge wird beim automatischen Formelsatz verwendet.
Vorbesetzt: `\medmuskip=4mu plus 2mu minus 4mu`

`\medskip` Textsatz: beendet einen Absatz und erzeugt einen vertikalen Leerraum im Umfang von `\medskipamount`. Vorbelegt ist dies mit einer halben Zeilenhöhe 27
(siehe auch `\smallskip`, `\bigskip`).
D
```
\def\medskip{\vskip \medskipamount }
```

`\medskipamount` Textsatz: vertikaler Leerraum, der durch `\medskip` gesetzt wird.
D
```
\newskip\medskipamount
\medskipamount=6pt plus 2pt minus 2pt
```

∗ `\message` erzeugt eine Hilfsausgabe auf dem Terminal bzw. in die Protokolldatei.
Aufruf: `\message{..information..}`

`\mid` Mathematiksatz: Relation — | — 86
D
```
\mathchardef\mid="326A
```

`\midinsert` Befehl zur Einfügung von Information an der aktuellen Position; ist nicht mehr genügend Platz vorhanden, so wird die angegebene Information an den Anfang der folgenden Seite gesetzt. 49
Syntax: `\midinsert` *vertikales Material* `\endinsert`
(siehe auch `\topinsert` und `\pageinsert`)
D `\def\midinsert{\@midtrue \@ins }`

`\min` Mathematiksatz: großer Operator — min — 78
D `\def\min{\mathop {\rm min}}`

`minus` Schlüsselwort bei Längenangaben für *skip*-Information, und zwar wird der 'Schrumpfanteil' angegeben. *29*
Nach `\hskip 1cm minus 1cm` darf der horizontale Leerraum auch auf 0,5 cm reduziert werden, wenn nicht mehr genügend Platz vorhanden ist
(siehe auch `plus`).

`\mit` Mathematiksatz: Anwahl "math italic Schriftfamilie" 58
D `\def\mit{\fam \@ne }`

`mm` Maßeinheit Millimeter. 22
1 mm ≈ 2,854 pt — 1 pt ≈ 0,351 mm

∗ `\mkern` Mathematiksatz: Dies ist das Äquivalent zum `\kern`-Befehl im Textsatz. `\mkern` darf nur mit der Maßeinheit `mu` *(math units)* verwendet werden. Je nach Modus wird vertikaler oder horizontaler Leerraum erzeugt. *154*
18 mu = 1 em
(siehe auch `\thinmuskip`, `\medmuskip`, `\thickmuskip`)

`\models` Mathematiksatz: Relation — ⊨ — 86
D `\def\models{\mathrel |\joinrel =}`

∗ `\month` internes Register, wird beim Start mit dem Monat des Tagesdatums besetzt. Ausgabe mit `\the\month`
(siehe auch `\day`, `\year`, `\time`).

∗ `\moveleft` Eine Box wird damit im vertikalen Modus, wenn Elemente untereinander angeordnet werden, nach links verschoben. 144
Beispiel:

```
\hbox{\vrule \vbox{\hbox{AAAAA}%
                   \moveleft8pt\hbox{BBBBB}%
                   \hbox{CCCCC}}%
      \vrule}
```

ergibt

AAAAA
BBBBB
CCCCC

(siehe auch `\moveright`, `\raise`, `\lower`).

∗ `\moveright` Eine Box wird damit im vertikalen Modus, wenn Elemente untereinander angeordnet werden, nach rechts verschoben (siehe auch `\moveleft`, `\raise`, `\lower`). 144

`\mp` Mathematiksatz: binärer Operator — ∓ — (siehe auch `\pm` für — ± —) 86
D `\mathchardef\mp="2207`

`\mscount` internes `\count` Register für die plain-TEX Makros. Es wird im plain-TEX-Makro `\multispan` verwendet.
D `\newcount\mscount`

∗ `\mskip` Mathematiksatz: fügt Leerraum im Mathematiksatz ein. Die Einheit dafür ist `mu` *(math units)*. So ist etwa `\,` als `\def\,{\mskip\thinmuskip}` definiert.

`mu` Maßeinheit für Dimensionen im Mathematiksatz: 18 mu = 1 em

`\mu` Mathematiksatz: griechischer Buchstabe — μ — 66
D `\mathchardef\mu="116`

∗ `\multiply` allgemeiner Multiplikationsbefehl für die Muplikation eines Registers mit einer ganzen, auch negativen Zahl.
Beispiel:
```
\multiply\count3 by 5
\multiply\bigskipamount by 2
```
Da vor alle `\dimen`- und `\skip`-Register ein Faktor geschrieben werden kann, wird dieser Befehl nur selten benutzt.

`\multispan` Tabellensatz (Text- und Mathematiksatz): 103, *107*
Wird zur Ausgabe von Tabellenelementen verwendet, wenn mehrere Spalten als ein Element betrachtet werden sollen.
Beispiel:
```
\halign{\hfill#\hfill\quad&
        \hfill#\hfill\quad&
        \hfill#\hfill\quad&
        \hfill#\hfill\quad&#\cr
1 & 2  &  3  & 4  \cr
5 & \multispan2\hfill total \hfill& 6  \cr
7 & 8  &  9  & 10 \cr}
```

1	2	3	4
5		total	6
7	8	9	10

D
```
\def\multispan#1{\omit \mscount #1
      \loop \ifnum \mscount >\@ne \sp@n \repeat }
```

∗ `\muskip` Mathematiksatz: Damit kann eins von 256 *muskip*-Registern referiert werden 147
z.B. `\muskip17=1mu  \mskip\muskip`.
Durch `\newmuskip\meinemu` kann man sich ein freies Register besorgen und damit arbeiten.

∗ `\muskipdef` interner Befehl zur Benennung von *muskip*-Registern.
Durch `\muskipdef\meinemu=17` wird Register 17 der Name `\meinemu` zugeordnet.
Dann darf man auch `\muskip\meinemu` eingeben.

`\nabla` Mathematiksatz: normales Zeichen — ∇ — 88
D `\mathchardef\nabla="272`

`\narrower` Textsatz: erhöht die Register `\leftskip` und `\rightskip` um `\parindent`. Die Absätze werden schmaler gesetzt. Wird der Befehl mehrfach aufgerufen, wird der Absatz immer schmaler. 36
D
```
\def\narrower{\advance \leftskip \parindent
              \advance \rightskip \parindent }
```

`\natural` Mathematiksatz: normales Zeichen — ♮ — 88
(siehe auch `\flat` — ♭ —)
D `\mathchardef\natural="15C`

`\ne` Mathematiksatz: Relation — ≠ — 86
(äquivalent mit `\neq`)
D `\def\ne{\not =}`

`\nearrow` Mathematiksatz: Relation — ↗ — 87
(north east arrow)
D `\mathchardef\nearrow="3225`

`\neg` Mathematiksatz: normales Zeichen — ¬ — 88
(äquivalent zu `\lnot`)
D `\mathchardef\neg="23A`

`\negthinspace` erzeugt einen kleinen Rückwärtsskip im Umfang von `\thinspace`: || 29
D `\def\negthinspace{\kern -.16667em }`

`\neq` Mathematiksatz: Relation — ≠ — 86
(äquivalent `\ne` und `\not=`)
D `\def\neq{\not =}`

`\newbox` Reserviert das nächste freie Box-Register unter dem gewünschten Namen. Nach `\newbox\meinebox` steht in `\meinebox` die *Nummer* dieses Registers. Damit kann diese in Befehlen wie `\setbox\meinebox=\hbox{...}` oder `\copy\meinebox` usw. verwendet werden. 132
(Anmerkung: Es gibt *keinen* `\boxdef`-Befehl.)
D `\outer \def\newbox{\alloc@ 4\box \chardef \insc@unt }`

`\newcount` Reserviert das nächste freie *count*-Register und gibt ihm den Namen, der als Parameter folgt. Damit kann es dann in weiteren Befehlen unter diesem Namen angesprochen werden. 147

Beispiel:

```
\newcount\zaehler
\zaehler=13
\advance\pageno by \zaehler
```

```
\outer \def\newcount{\alloc@ 0\count
                     \countdef \insc@unt }
```

`\newdimen` Reserviert das nächste freie *dimen*-Register und gibt ihm den Namen, der als Parameter folgt. Damit kann es dann in weiteren Befehlen unter diesem Namen zur Speicherung oder zum Abruf von Längenangaben verwendet werden. 147, *156*

Beispiel:

```
\newdimen\laenge
\laenge=4cm
\advance\leftskip by \laenge
\rightskip=0.3\laenge
```

```
\outer \def\newdimen{\alloc@ 1\dimen
                     \dimendef \insc@unt }
```

`\newfam` Reserviert die nächste freie Nummer einer der 16 möglichen Schriftfamilien — bisher sind belegt:

0	`\rm (\rmfam)`	'roman' Schrift
1	`\mit`	math italic fonts
2	`\cal`	math symbol fonts
3		math extension fonts
4	`\it (\itfam)`	*'italic'* Schrift
5	`\sl (\slfam)`	*'slanted'* Schrift
6	`\bf (\bffam)`	**'boldface'** Schrift
7	`\tt (\ttfam)`	`'typewriter'` Schrift

Jede Schriftfamilie besitzt `\textfont`, `\scriptfont` und `\scriptscriptfont` Angaben, die die Schriften für die verschiedenen Modi des mathematischen Satzes definieren. Dies sind entsprechend die Satzweisen im *text-style, script-style* oder *scriptscript-style*. Typischerweise sind dann jeweils Schriften in 10, 7 und 5-Punkt Größe zugeordnet.

(So ist etwa definiert: `\textfont0=\tenrm`)

TEX verwendet die ersten Schriftfamilien 0 bis 3 für den mathematischen Formelsatz. So werden beispielsweise die Fontparameter `\fontdimen` aus den Schriftfamilien 2 und 3 zur Satzsteuerung für Indizes und Exponenten verwendet.

```
\outer \def\newfam{\alloc@ 8\fam
                   \chardef \sixt@@n }
```

`\newif` erzeugt eine "neue" *if*-Anweisungen. 117
Beispiel: Nach `\newif\ifsecret` stehen folgende Befehle zur Verfügung:
`\secrettrue` setzt `\ifsecret` auf *true*,
`\secretfalse` setzt `\ifsecret` auf *false*,
`\ifsecret` fragt den aktuellen Wert ab.
Der neue *if*-Befehle muß mit "`\if`" beginnen, dies wird durch `\newif` gegrüft.

```
\outer \def\newif#1{\count@ \escapechar
    \escapechar \m@ne
    \expandafter \expandafter \expandafter
      \edef \@if #1{true}{%
            \let \noexpand #1=\noexpand \iftrue}%
    \expandafter \expandafter \expandafter
      \edef \@if #1{false}{%
            \let \noexpand #1=\noexpand \iffalse }%
    \@if #1{false}%
    \escapechar \count@ }
```

D

`\newinsert` reserviert eine *insertion*-Registerfolge.
Durch beispielsweise `\newinsert\myins` wird für `\myins` die Nummer des *insertion*-Registers und der dazugehörigen Arbeitsregister festgelegt. Das heißt, für folgendes Beispiel sind dann:
`\box\myins` die Box, wo die Information während der *output-routine* erscheint,
`\count\myins` der Skalierungsfaktor mit dem der Anteil berechnet wird, der durch eine gespeicherte Einfügung von der aktuellen Seite abgezogen werden soll,
`\dimen\myins` die maximale Einfügung je Seite,
`\skip\myins` der extra Platz, der für die erste Einfügung je Seite reserviert werden soll.

Auf diese Weise werden zum Beispiel Fußnoten und Einfügungen mittels `\topinsert` verarbeitet.

```
\outer\def\newinsert#1{%
  \global\advance\insc@unt by\m@ne
  \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}}
```

D

3 `\newlanguage` benennt etwa mittels `\newlanguage\German` durch Vergabe der nächsten freien Sprachnummer eine Sprache. Dies wird nur in INITEX vollzogen.

* `\newlinechar` Internes Register, das den ASCII-Code für das Zeilenende einer Eingabezeile enthält. Normalerweise ist dies mit 13 für 'CR' im ASCII-Code vorbesetzt.
Genau genommen ist dies das Zeichen, das TEX in die Eingabe einfügt, wenn durch das Programm ein Zeilenende erkannt wird. Dies geschieht auch beim `\read` Befehl.

`\newmuskip` reserviert das nächste freie *muskip*-Register unter diesem Namen. Damit kann es dann in weiteren Befehlen unter diesem Namen referiert werden. 147
Beispiel:
```
\newmuskip\mathskip
\mathskip=1.5mu
\advance\thickmuskip by \mathskip
\mskip\mathskip
```
D `\outer \def\newmuskip{% \alloc@ 3\muskip \muskipdef \@cclvi }`

`\newread` reserviert die nächste freie Nummer (0 . . . 15) für den Eingabestrom einer externen Datei. Durch 194
```
\newread\extrafile
\openin\extrafile=UUDATEN
\read\extrafile to \inputdata
```
wird die Datei "`UUDATEN.TEX`" eröffnet und der erste Satz auf `\inputdata` gelesen. Mehr als ein Satz wird gelesen, wenn die Klammerstruktur in der Eingabedatei dies verlangt.
D `\outer \def\newread{\alloc@ 6\read \chardef \sixt@@n}`

`\newskip` reserviert das nächste freie *skip*-Register unter dem folgenden Namen. Damit kann das Register dann in weiteren Befehlen unter diesem Namen verwendet werden. 147
Beispiel:
```
\newdimen\skipreg
\skipreg=1.5cm plus 0.5cm minus 0.5cm
\hskip\skipreg
```
D `\outer \def\newskip{% \alloc@ 2\skip \skipdef \insc@unt }`

`\newtoks` reserviert das nächste freie *token*-Register. 148
Beispiel:

`\newtoks\toktok`	neues Register
`\toktok={...data...}`	Zuweisung
`\the\toktok`	Ausgabe
`\showthe\toktok`	Protokoll

D `\outer \def\newtoks{\alloc@ 5\toks \toksdef \@cclvi }`

`\newwrite` entspricht (`\newread`) nur für Ausgabedateien. 194, *196*
Beispiel:

```
\newwrite\myoutfile              Reservierung
\openout\myoutfile=AUSGABE       Öffnung
\write\myoutfile{...data...}     Schreiben
\closeout\myoutfile              Schließen
```

D `\outer \def\newwrite{% \alloc@ 7\write \chardef \sixt@@n }`

`\next` ist ein in anderen Makros häufig verwendetes lokales Hilfsmakro. Es wird stets neu definiert.
Vorbelegt: `\let\next\relax`

`\ni` Mathematiksatz: Relation — ∋ — 86
(äquivalent ist `\owns`)
D `\mathchardef\ni="3233`

* `\noalign` Tabellensatz (in `\halign`, `\matrix` ...) *81*, 96, 104
`\noalign{` *vertikales Material* `}`
setzt die angegebene Information zwischen die einzelnen Tabellenzeilen.
Beispiel:
`\noalign{\smallskip}` setzt einen zusätzlichen Leerraum zwischen zwei Tabellenzeilen.

3 * `\noboundary` Dieser Befehl wird entweder am Anfang oder am Ende eines Wortes gegeben. Er sorgt dann intern dafür, daß der Wortanfang oder das Wortende nicht mehr als Anfang oder Ende interpretiert werden. Dies ist dann von Bedeutung, wenn Ligatureinträge in der jeweiligen Sprache eingetragen sind, die für Wortanfänge und Wortenden besondere Einträge besitzen. (Beispielsweise gibt es im Deutschen Schriften, bei denen der Buchstabe 's' am Wortende unterschiedlich dargestellt wird.)

`\nobreak` verhindert Zeilenumbruch oder Seitenwechsel je nach Position durch Erzeugung von 10000 Minuspunkten. 45, 46
D `\def\nobreak{\penalty \@M }`

* `\noexpand` verhindert die Expandierung eines nachfolgenden Makroaufrufes *(token)*. Die typische Anwendung ist im Zusammenhang mit `\edef` *(expanded definition)*. 113, 114, *198*
Bei `\edef` werden alle Makros, die im definierenden Text stehen, schon bei der Definition vollständig expandiert — es sei denn `\noexpand` steht vor einem Makroaufruf.
Beispiel:
`\def\a{aaa}  \edef\b{\a\noexpand\a}`
ist äquivalent zu der Definition
`\def\b{aaa\a}`
Wird `\b` hinterher aufgerufen, so wird stets die aktuelle Bedeutung von `\a` eingesetzt!

∗ `\noindent` Hiermit wird nach einer Leerzeile oder einem `\par` ein neuer Absatz ohne Einzug in der ersten Zeile begonnen (siehe auch `\indent`, `\parindent`). 21

`\nointerlineskip` 145, *173*

verhindert *einmalig*, daß der Leerplatz zwischen zwei untereinander gesetzten Boxen (im *vertical mode*) gemäß `\baselineskip`, `\lineskip` und `\lineskiplimit` gesetzt wird. Die Boxen können also direkt anstoßen oder sich sogar überlappen.
Dies geschieht, indem die logische Unterlänge *(Tiefe, 'depth')* der vorangehenden Box durch den Befehl

```
\prevdepth=-1000pt
```

reduziert wird (siehe auch `\offinterlineskip`).

D `\def\nointerlineskip{\prevdepth -1000\p@ }`

∗ `\nolimits` Mathematiksatz: Folgt dieser Befehl einem 'großen Operator', dann werden Exponent- und Indexteile stets neben das Symbolzeichen gesetzt. 74, *78*
Beispiel:
`$$\sum\nolimits_{i=1}^\infty$$`

führt zu $\sum_{i=1}^{\infty}$ — normal: $\sum\limits_{i=1}^{\infty}$

(siehe auch `\limits`, `\displaylimits`)

`\nonfrenchspacing` 16

Textsatz: schaltet in den voreingestellten Satzmodus für die Leerraumbehandlung nach Satzzeichen zurück. Dadurch wird nach Satzzeichen etwas mehr Leerraum gesetzt. Die andere Einstellung wird durch den Befehl `\frenchspacing` erreicht.

D
```
\def\nonfrenchspacing{\sfcode `\.3000\sfcode `\?3000%
    \sfcode `\!3000\sfcode `\:2000%
    \sfcode `\;1500\sfcode `\,1250 }
```

∗ `\nonscript` Mathematiksatz: Wird der Befehl `\nonscript` einem *skip*-Befehl im Mathematiksatz vorangestellt, so wird dieser im *script-style* und im *scriptscriptstyle* nicht ausgeführt. (Dies erspart eine `\mathchoice`-Operation.)

∗ `\nonstopmode` setzt den *non-stop-mode*. Daraufhin hält das Programm im Fehlerfall nicht mehr für Anfragen an. Die Fehlermeldungen werden im interaktiven Betrieb aber noch auf dem Terminal protokolliert. 164
Dieser Modus wird auch durch Eingabe von "`R`" bei einer Fehleranfrage erreicht.

`\nopagenumbers` Textsatz: Im weiteren werden keine Seitenfußzeilen mehr ausgegeben. Es erscheint also auch keine Seitennumerierung im Standardformat mehr. 47

D `\def\nopagenumbers{\footline {\hfil }}`

`\normalbaselines`

Textsatz: Restauriert die Werte von `\lineskip`, `\lineskiplimit` und `\baselineskip` durch die gespeicherten Werte in den Registern

`\normallineskip` 1 pt
`\normallineskiplimit` 0 pt
`\normalbaselineskip` 12 pt.

```
D  \def\normalbaselines{\lineskip \normallineskip
               \baselineskip \normalbaselineskip
               \lineskiplimit \normallineskiplimit }
```

`\normalbaselineskip`

plain-TEX Register zur Speicherung des Standardzeilenabstandes (12 pt), dieser wird beim Befehl `\normalbaselines` dem Register `\baselineskip` zugewiesen.

```
D  \newskip\normalbaselineskip \normalbaselineskip=12pt
```

`\normalbottom` Textsatz: stellt den "flatternden Seitenspiegel" wieder auf den Normalzustand zurück. Durch `\raggedbottom` kann ja eine in gewissem Umfang unterschiedliche Seitenlänge erlaubt werden.

```
D  \def\normalbottom{\topskip 10\p@ \r@ggedbottomfalse }
```

`\normallineskip`

plain-TEX Register zur Speicherung des Minimalabstandes (1 pt) zwischen untereinanderstehenden Boxen. Dieser wird beim Befehl `\normalbaselines` dem Register `\lineskip` zugewiesen.

```
D  \newskip\normallineskip \normallineskip=1pt
```

`\normallineskiplimit`

plain-TEX Register zur Speicherung des Prüfwertes (0 pt) für den Abstand untereinander stehenden Boxen; dieser wird beim Befehl `\normalbaselines` dem internen Register `\lineskiplimit` zugewiesen.

```
D  \newdimen\normallineskiplimit
   \normallineskiplimit=0pt
```

`\not` Mathematiksatz: Wird der Befehl `\not` einem Befehl für eine Relation vorangestellt, so wird diese in verneinter Form ausgegeben. Diese wird mit einem Schrägstrich durchstrichen dargestellt. De facto ist `\not` nichts weiter als ein Schrägstrich mit der logischen Breite von Null, der das folgende Zeichen überdruckt. 86

Beispiel:

`\not=` liefert — $\neq$ —
`\not\sim` liefert — $\not\sim$ —

```
D  \mathchardef\not="3236
```

`\notin` Mathematiksatz: Relation — $\notin$ — 87
(fast äquivalent zu `\not\in`, nur sitzt der Schrägstrich weiter links)
D `\def\notin{\mathrel {\mathpalette \c@ncel \in }}`

`\nu` Mathematiksatz: griechischer Buchstabe — ν — 66
D `\mathchardef\nu="117`

`\null` erzeugt eine leere `\hbox`.
D `\def\null{\hbox{}}`

* `\nulldelimiterspace`
Mathematiksatz: Platz, den ein leerer Delimiter `\left.` oder `\right.` einnimmt.
Vorbesetzt: `\nulldelimiterspace=1.2pt`

* `\nullfont` Name des internen leeren Fonts, der immer dann automatisch substituiert wird, wenn die mittels `\font` angewählte Schrift nicht gefunden wird. Abhängig von der Belegung von `\tracinglostchars` wird dann die Verwendung eines Zeichens aus `\nullfont` protokolliert.

* `\number` gibt eine Zahl oder den Inhalt eines `\count`-Register aus.
Beispiel:

`\number24`	ergibt — 24 —
`\number-0010`	ergibt — -10 —
`\count7=18`	
`\number\count7`	ergibt — 18 —

(siehe auch `\romannumeral`)

`\nwarrow` Mathematiksatz: Relation — $\nwarrow$ — 87
(north west arrow)
D `\mathchardef\nwarrow="322D`

`\o` Textsatz: liefert — ø — 25
D `\chardef\o="1C`

`\O` Textsatz: liefert — Ø — 25
D `\chardef\O="1F`

`\oalign` Textsatz: Hilfsmakro zur Erzeugen der *Cedille* mittels `\c` sowie weiterer Akzente `\d`, `\b`.
D
```
\def\oalign#1{\leavevmode \vtop{\baselineskip \z@skip
        \lineskip .25ex\ialign {##\crcr #1\crcr }}}
```

`\obeylines` Textsatz: stellt die Interpretation des Zeilenendes auf einen impliziten `\par`-Befehl um. Dadurch wird jede Eingabezeile zu einem Absatz für sich, die dann allerdings auch eine Absatzeinrückung der Größe `\parindent` besitzt. Eine zeilenweise Eingabe nicht zu langer Zeilen wird dann auch zeilenweise ausgegeben. 33ff

Beispiel:

```
{\obeylines \it
  erste Zeile
    zweite Zeile
  dritte Zeile\par}
```

wird gesetzt als:

erste Zeile
zweite Zeile
dritte Zeile

D `\def\obeylines{\catcode `\^^M\active \let ^^M\par }`

`\obeyspaces` Textsatz: Nach `\obeyspaces` werden *alle* Leerzeichen auch mehrere hintereinander für sich ausgegeben.

D
```
\def\obeyspaces{\catcode `\ \active }
{\obeyspaces\global\let =\space}
```

`\odot` Mathematiksatz: binärer Operator — ⊙ — 86

D `\mathchardef\odot="220C`

`\oe` Textsatz: liefert — œ — 25, 25, 25

D `\chardef\oe="1B`

`\of` ist ein Schlüsselwort beim `\root` Befehl. 69

`\OE` Textsatz: liefert — Œ — 25, 25

D `\chardef\OE="1E`

`\offinterlineskip` 102ff, 145

schaltet jeglichen Zeilenzwischenraum zwischen Zeilen und im *vertical mode* untereinander gesetzten Boxen ab: (Durch den Befehl `\normalbaselines` werden die Parameter wieder auf die Standardeinstellung in plain-TEX (12 pt, 1 pt,0 pt) zurückgestellt.)

D
```
\def\offinterlineskip{\baselineskip -1000\p@
                      \lineskip \z@
                      \lineskiplimit \maxdimen }
```

`\oint` Mathematiksatz: großer Operator — ∮ — 73

D `\def\oint{\ointop \nolimits }`

`\oldstyle` Mathematiksatz: Liefert *oldstyle Ziffern*. 88

`$\oldstyle 0123456789$` liefert — 0123456789 —

D `\def\oldstyle{\fam \@ne \teni }`

`\omega` Mathematiksatz: griechischer Buchstabe — ω — 66

D `\mathchardef\omega="121`

\Omega — Mathematiksatz: griechischer Buchstabe — Ω — 67
D \mathchardef\Omega="700A

\ominus — Mathematiksatz: binärer Operator — ⊖ — 86
D \mathchardef\ominus="2209

∗ \omit — Tabellensatz mit \halign: Ein \omit-Befehl am Anfang eines Tabelleneintrags unterdrückt die Auswertung der Musterzeile für dieses Element. Das heißt, das Musterelement wird auf "#" reduziert. 103, *107*

\ooalign — Hilfsmakro zum Setzen von \c und \copyright.
D \def\ooalign{\lineskiplimit -\maxdimen \oalign }

∗ \openin — öffnet einen von 16 möglichen zusätzlichen Eingabeströmen für eine externe Datei (0 . . . 15). Statt der direkten Zahlangabe sollte besser ein mittels \newread definierte Name verwendet werden. 194
Beispiel:
```
\openin 7 = MYFILE
\read 7 to \meintext
```
eröffnet die Eingabedatei "MYFILE.TEX" unter der Nummer 7. Durch \read... wird ein Satz nach \meintext gelesen
(siehe auch \read, \closein, \newread, \ifeof).

∗ \openout — öffnet einen von 16 möglichen zusätzlichen Ausgabeströmen für eine externe Datei. Statt der direkten Eingabe einer Zahl sollten mittels \newwrite definierte Namen verwendet werden. 194, *196*
Ohne ein vorangestelltes \immediate wird \openout erst in der *output routine* ausgeführt!
Beispiel:
```
\openout 7 = MYFILE
\write 7 {...information...}
```
eröffnet die Ausgabedatei "MYFILE.TEX" unter der Nummer 7. Durch \write.. wird ein Satz ausgegeben
(siehe auch \write, \closeout, \immediate).

\openup — vergrößert die drei Kontrollparameter für den Zeilenabstand \baselineskip, \lineskip und \lineskiplimit um die angegebene Länge. So wird in einigen Mathematikmakros \openup3pt verwendet, um mehrzeilige Formeln mit größerem Abstand zu setzen.
```
\def\openup{\afterassignment \@penup \dimen@ =}
\def\@penup{\advance \lineskip \dimen@
            \advance \baselineskip \dimen@
            \advance \lineskiplimit \dimen@}
```
D

\oplus — Mathematiksatz: binärer Operator — ⊕ — 86
D \mathchardef\oplus="2208

∗ `\or` Schlüsselwort beim `\ifcase`-Befehl
Syntax:

```
\ifcase Zahl oder Zahlregister
        Befehle für 0
  \or   Befehle für 1
  \or   Befehle für 2
    ...
  \else Befehle für sonstige Fälle
\fi
```

`\oslash` Mathematiksatz: binärer Operator — ⊘ — 86
D `\mathchardef\oslash="220B`

`\otimes` Mathematiksatz: binärer Operator — ⊗ — 86
D `\mathchardef\otimes="220A`

∗ `\outer` Schlüsselwort, das einem `\def`, `\edef` oder `\gdef` vorangestellt werden kann. Es bewirkt, daß das so definierte Makro nur *außen* aufgerufen werden kann, also nicht als Makroparameter oder in einer Box. So ist "`\bye`" als "`\outer`" deklariert. Das Makro `\outertest`, definiert durch `\outer\def\outertest{}`, bewirkt eine Fehlermeldung "`Forbidden control sequence found ..`", wenn es innerhalb einer Makrodefinition verwendet wird. *35*, 114

∗ `\output` ist das *token*-Register, welches den Verweis auf die eingestellte *ouput-routine* enthält. In plain-TEX ist es belegt durch 172ff

```
\output={\plainoutput}
```

Demnach wird die Routine `\plainoutput` aufgerufen, wenn eine gemäß `\vsize` gefüllte Seite festgestellt wird.

∗ `\outputpenalty` enthält beim Aufruf der *output-routine* die Bewertung *(penalty)* der gewählten Umbruchstelle; dies sind die Minuspunkte für den Seitenumbruch. Durch eine geeignet geschaffene *output-routine* ist es dann möglich, mittels Iteration den Umbruch und daraus folgend diesen Wert zu verändern. 171

∗ `\over` Mathematiksatz: Standardbefehl zum Satz eines Bruches. 71
Links von "`\over`" steht der Zähler, rechts der Nenner.
Beispiel:

```
$$ 1 \over { 2 \over x + 3 } + 1 $$
```

ergibt

$$\frac{1}{\frac{2}{x+3}+1}$$

(siehe auch `\above` und `\atop`)

`\overbrace` Mathematiksatz: Es wird eine geschweifte Klammer über einer Formel erzeugt. Beispiel: `$$ \overbrace{a + \cdots + a}^{\hbox{n mal}}$$` erzeugt 83

$$\overbrace{a + \cdots + a}^{\text{n mal}}$$

(siehe auch `\underbrace`; für den Textsatz: `\downbracefill`, `\upbracefill`)

```
\def\overbrace#1{\mathop
   {\vbox{\ialign {##\crcr
          \noalign{\kern 3\p@ } \downbracefill \crcr
          \noalign{\kern 3\p@ \nointerlineskip }
          $\hfil \displaystyle {#1}\hfil $\crcr
   }}}\limits }
```

D

∗ `\overfullrule` Strichdicke des Markierungsbalkens bei der Ausgabe zur Kennzeichnung überfüllter Zeilen oder Boxen. Eine Box gilt als überfüllt, wenn sie um mehr als `\hfuzz` breiter ist als sie sein dürfte. Ist `\hfuzz` größer als Null, so werden kleinere Überfüllungen toleriert und auch nicht markiert. Durch `\overfullrule=0pt` wird die Fehlermarkierung vollständig unterdrückt. 42, 54

`\overleftarrow` Mathematiksatz: zieht einen Pfeil über die als Parameter angegebene Formel. Beispiel: `$\overleftarrow{A-B}$` erzeugt $\overleftarrow{A-B}$ (siehe auch `\overrightarrow`) 70

```
\def\overleftarrow#1{\vbox
   {\ialign {##\crcr \leftarrowfill \crcr
       \noalign {\kern -\p@ \nointerlineskip }
       $\hfil \displaystyle {#1}\hfil $\crcr }}}
```

D

∗ `\overline` Mathematiksatz: zieht eine Linie über die als Parameter angegebene Formel. Beispiel: `$\overline{A/B}$` erzeugt $\overline{A/B}$ (siehe auch `\underline`) 69

`\overrightarrow` 70

Mathematiksatz: zieht einen Pfeil über die als Parameter angegebene Formel. Beispiel: `$\overrightarrow{A-B}$` erzeugt$\overrightarrow{A-B}$ (siehe auch `\overleftarrow`)

```
\def\overrightarrow#1{\vbox {\ialign {##\crcr
       \rightarrowfill \crcr
       \noalign {\kern -\p@ \nointerlineskip }
       $\hfil \displaystyle {#1}\hfil$\crcr }}}
```

D

∗ `\overwithdelims` 77

Mathematiksatz: Dieser Befehl erzeugt einen Bruch mit normaler Bruchstrichdicke, wobei angegeben wird, welche automatisch wachsenden Klammern links und rechts vom Bruch stehen sollen.
Beispiel:
`$$ f(G) \overwithdelims<> h(G) $$`
erzeugt

$$\left\langle \frac{f(G)}{h(G)} \right\rangle$$

(siehe auch `\abovewithdelims`, `\atopwithdelims`)

`\owns` Mathematiksatz: Relation — ∋ — 86
(äquivalent mit `\ni`)
D `\mathchardef\owns="3233`

`\P` Textsatz: liefert — ¶ — 88
(paragraph/pilcrow)
D `\def\P{\mathhexbox 27B}`

`\pagebody` ist ein plain-TEX Hilfsmakro für die *output-routine*. 173
D
```
\def\pagebody{\vbox to\vsize {\boxmaxdepth \maxdepth
        \pagecontents }}
```

`\pagecontents` ist ein plain-TEX Hilfsmakro für die *output-routine*. 173
D
```
\def\pagecontents{%
    \ifvoid \topins \else \unvbox \topins \fi
    \dimen@ =\dp \@cclv
    \unvbox \@cclv
    \ifvoid \footins
    \else \vskip \skip \footins \footnoterule
          \unvbox \footins \fi
    \ifr@ggedbottom \kern -\dimen@ \vfil \fi }
```

∗ `\pagedepth` Register mit der Unterlänge *"depth"* der aktuellen Seite.

∗ `\pagefilllstretch`
ist ein internes Register für den Seitenumbruch. Es enthält den akkumulierten *fillll*-Anteil auf der aktuellen Seite (in `\filll`-Einheiten).

∗ `\pagefillstretch`
ist ein internes Register für den Seitenumbruch. Es enthält den akkumulierten *fill*-Anteil auf der aktuellen Seite (in `\fill`-Einheiten).

∗ `\pagefilstretch`
ist ein internes Register für den Seitenumbruch. Es enthält den akkumulierten *fil*-Anteil auf der aktuellen Seite (in `\fil`-Einheiten).

*	\pagegoal	enthält die zu erzielende aktuelle Seitenhöhe für den Seitenumbruch. Der Wert wird mit \vsize initialisiert. Bei Fußnoten und Einfügungen wird er jeweils reduziert. Die Differenz \pagegoal - \pagetotal ist dann der noch freie Raum auf einer Seite (ohne Berücksichtigung der dynamischen Anteile).	
	\pageinsert	Textsatz: Dieser Befehl tritt nur in der Kombination \pageinsert ... \endinsert auf. Die zwischen den beiden Befehlen angegebene Information wird als eigene Seite ausgegeben (siehe auch \midinsert, \topinsert). D \def\pageinsert{\@midfalse \p@getrue \@ins }	49
	\pageno	ist das Register mit der aktuellen Seitennummer. (Benennung des Registers \count0) Durch \pageno=17 kann zum Beispiel das Register verändert werden, beim Seitenwechsel wird es automatisch weitergezählt. Zur Ausgabe wird es mittels \folio referiert. D \countdef\pageno=0 \pageno=1	46ff
*	\pageshrink	ist ein internes Register für den Seitenumbruch. Es enthält den akkumulierten Schrumpfanteil auf der aktuellen Seite; das heißt die Länge, um die der Leerraum auf der aktuellen Seite bei Bedarf noch zusammengeschoben werden kann.	
*	\pagestretch	ist ein internes Register für den Seitenumbruch. Es enthält den akkumulierten Zuwachsanteil auf der aktuellen Seite; das heißt die Länge, um die der Leerraum auf der aktuellen Seite bei Bedarf noch auseinander gezogen werden kann.	
*	\pagetotal	ist ein internes Register für den Seitenumbruch. Es enthält den Umfang der bisher gebildeten Seite.	
*	\par	Textsatz: beendet einen laufenden Absatz.	21, 22
	\parallel	Mathematiksatz: Relation — ‖ — (als normales Zeichen unter \Vert oder |) D \mathchardef\parallel="326B	86
*	\parfillskip	Textsatz: Horizontaler *skip* am Absatzende. Durch seine Vorbesetzung mit \parfillskip=0pt plus 1fil wird automatisch dafür gesorgt, daß die letzte Zeile eines Absatzes linksbündig steht und nicht zum rechten Rand hin ausgeglichen werden muß.	34
*	\parindent	Textsatz: Größe des Einzuges zu Beginn eines Absatzes bei \indent und \item. Vorbesetzt: \parindent=20pt (Diese Größe wird auch von \narrower benutzt.)	21, *43*, *45*, *51*

∗ \parshape — Textsatz: definiert für einen einzelnen Absatz Zeile für Zeile dessen Aussehen.
Syntax: \parshape = $n\ i_1\ l_1\ \ldots\ i_n\ l_n$
Dabei ist n die Anzahl Zeilen, i_j die linke Einrückung für die j-te Zeile und l_j die Restlänge dieser Zeile. — 40, *40*

\partial — Mathematiksatz: normales Zeichen – ∂ — 88
D \mathchardef\partial="140

∗ \parskip — Textsatz: Dies ist der Abstand, der automatisch zwischen einzelnen Absätzen erzeugt wird.
Vorbesetzt: \parskip=0pt plus 1pt — 27, 38

∗ \patterns — Dieser Befehl ist nur in INITEX vorhanden. Er dient zur Eingabe von Trennmustern für die Silbentrennung. Dabei werden die Trennmuster der Sprache zugeordnet, die durch den aktuellen Wert von \language definiert ist.

∗ \pausing — ist eine interne mit Null initialisierte Steuervariable. Nach der Umsetzung durch \pausing=1 hält TEX nach dem Einlesen einer Eingabezeile aus einer Datei an und legt diese zur Quittierung oder Ersetzung durch anderen Text vor. Dies geschieht vor der Interpretation dieser Zeile. Wird dann eine Eingabe gemacht, die nicht nur aus Leerzeichen besteht, so ersetzt diese die vorgelegte Zeile vollständig. Man beachte allerdings, daß dabei die Eingabedatei selbst *nicht* verändert wird.

∗ \penalty — ist der allgemeine Befehl, um Minuspunkte (oder auch Pluspunkte) abzusetzen. Damit wird der Zeilen- und Seitenumbruch beeinflußt. Bei positiven Zahlen sind es Minuspunkte, bei negativen Pluspunkte. Also wird durch \penalty-100 eine Umbruchstelle als 'gut' gekennzeichnet (siehe auch \break, \nobreak, ~). — 52

\perp — Mathematiksatz: Relation — $\perp$ — 86
D \mathchardef\perp="323F

\phantom — setzt die in angegebene Information nicht, sondern läßt genau so viel Platz in Breite, Höhe und Tiefe, wie sie bei einer normalen Ausgabe einnehmen würde
(siehe auch \hphantom, \vphantom, \smash).
D \def — *29*, 88, 89

\phi — Mathematiksatz: griechischer Buchstabe — ϕ —
(siehe auch \varphi für — φ —) — 66
D \mathchardef\phi="11E

\Phi — Mathematiksatz: griechischer Buchstabe — Φ — 67
D \mathchardef\Phi="7008

`\pi` Mathematiksatz: griechischer Buchstabe — π — (siehe auch `\varpi` für — ϖ —) 66

D `\mathchardef\pi="119`

`\Pi` Mathematiksatz: griechischer Buchstabe — Π — 67

D `\mathchardef\Pi="7005`

`\plainoutput` Standardmakro von plain-TEX für die *output-routine*. 172ff

```
D \def\plainoutput
    {\shipout \vbox {\makeheadline \pagebody
        \makefootline }\advancepageno
        \ifnum \outputpenalty >-\@MM
        \else \dosupereject \fi }
```

`plus` Schlüsselwort zur Angabe des wachsenden Anteils bei einem *skip*-Befehl *(glue)*. Dieser Anteil ist die Längenangabe für den Zuwachs, den eine Dimension etwa zur Auffüllung einer Box haben darf. 27, *29*

Beispiel:
`\vskip 1cm plus 0.5cm minus 0.5cm`
(siehe auch `minus`)

`\pm` Mathematiksatz: binärer Operator — $\pm$ — (siehe auch `\mp` für — $\mp$ —) 86

D `\mathchardef\pm="2206`

`\pmatrix` Mathematiksatz: *(parenthesized matrix)* gibt eine Matrix in Klammern aus. 79

Beispiel:
`$$\pmatrix{ 1 & 2 & 3 \cr 4 & 5 & 6 \cr 7 & 8 & 9 \cr}$$` liefert

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Beachte: Ganz intern wird mit `\halign` gesetzt
(siehe auch `\matrix`, `\bordermatrix`).

D `\def\pmatrix#1{\left (\matrix {#1}\right )}`

`\pmod` Mathematiksatz: *parenthesized modulo*

Beispiel:
`$ a \equiv b+1 \pmod m$` liefert

$$a \equiv b + 1 \pmod{m}$$

(siehe auch `\bmod`)

D `\def\pmod#1{\allowbreak \mkern 18mu({\rm mod}\,\,#1)}`

* `\postdisplaypenalty`
Minuspunkte für einen Seitenwechsel direkt nach einer Formel im *display-style*, das heißt einer in `$$...$$` eingeschlossenen Formel. Vorbelegt mit Null.

`\Pr` Mathematiksatz: großer Operator — Pr — 78
D `\def\Pr{\mathop {\rm Pr}}`

`\prec` Mathematiksatz: Relation — ≺ — 86
D `\mathchardef\prec="321E`

`\preceq` Mathematiksatz: Relation — ⪯ — 86
D `\mathchardef\preceq="3216`

* `\predisplaypenalty`
Minuspunkte für einen Seitenwechsel direkt vor einer Formel im *display-style*.
Vorbesetzt: `\predisplaypenalty=10000`

* `\predisplaysize`
Mathematiksatz: wird zu Beginn einer Formel im *display-style* automatisch mit der Länge der vorangehenden Zeile besetzt.

`\preloaded` Kennzeichnung von Fonts, die als 'preloaded' gelten, deren Daten geladen sind, aber noch einmal durch einen `\font`-Befehl neu definiert werden müssen, um sie verwenden zu können.

* `\pretolerance` Textsatz: interner Parameter zur Steuerung des Zeilenumbruchs. Er enthält die maximale Minuspunktzahl, die ein Absatz beim Umbruch haben darf, wobei diese für den ersten Durchgang ohne Trennungen gilt. 42, 53
Setzt man `\pretolerance=10000`, gilt der Absatz immer als akzeptiert; es wird also nicht getrennt
(siehe auch `\emergencystretch`, `\tolerance`).

* `\prevdepth` ist ein internes Register. Dies enthält die Unterlänge *(depth)* des vorangehenden Absatzes oder der vorangehenden Box im vertikalen Modus. Dieses Register kann auch umgesetzt werden; damit wird der Satz des folgenden Absatzes beeinflußt. Er kann so auch in den vorangehenden hineinragen
(siehe auch `\nointerlineskip`).

* `\prevgraf` Textsatz: interner Zähler. Er enthält die Anzahl der Zeilen, aus denen ein Absatz bis zu der aktuellen Position schon besteht. Der Wert ist aber nur dann belegt, wenn schon einige Zeilen fertig umbrochen sind, zum Beispiel vor einer hervorgehobenen Formel. (Veränderungen sind möglich. Sie werden aber nur im Zusammenhang mit `\hangafter` und `\parshape` ausgewertet.)

`\prime` Mathematiksatz: mathematischer Akzent. 69, 88
Beispiel:
```
$ { f(x)=x^2 }\to {f^\prime(x)=2x}$
```
ergibt

$$f(x) = x^2 \to f'(x) = 2x$$

Für `^\prime` kann auch ein einfaches Apostroph eingegeben werden: `$ ... f'(x)=2x} $`.

D `\mathchardef\prime="230`

`\prod` Mathematiksatz: großer Operator — $\prod$ — 73

D `\mathchardef\prod="1351`

`\proclaim` ist eigentlich ein Beispielmakro aus plain-TEX zum Setzen eines mathematischen Theorems oder einer Definition. Der erste Parameter wird durch einen Punkt, dem ein Leerzeichen folgt, der zweite durch `\par` abgetrennt. Nach
```
\proclaim Dilemma 1.
   Wer arbeitet macht Fehler.
```
wird die Ausgabe

Dilemma 1. *Wer arbeitet macht Fehler.*

mit etwas zusätzlichem Leerraum davor und dahinter erzeugt.

D
```
\outer\def\proclaim #1. #2\par{\medbreak
  \noindent{\bf#1.\enspace}{\sl#2\par}%
  \ifdim\lastskip<\medskipamount
       \removelastskip\penalty55\medskip\fi}
```

`\propto` Mathematiksatz: Relation — $\propto$ — 86

D `\mathchardef\propto="322F`

`\psi` Mathematiksatz: griechischer Buchstabe — ψ — 66

D `\mathchardef\psi="120`

`\Psi` Mathematiksatz: griechischer Buchstabe — Ψ — 67

D `\mathchardef\Psi="7009`

`pt` Maßeinheit *Druckerpunkt*. 22
1 pt ≈ 0,0351 cm — 1 cm ≈ 28,54 pt

`\qquad` Horizontaler Leerraum. 28
Dieser entspricht 2×`\quad` oder der Einheit 2 em.
Das ist die Breite von `\qquad`: | |

D `\def\qquad{\hskip 2em\relax }`

`\quad` horizontaler Leerraum. *16*, 28, *79*
Dieser entspricht 0.5×`\qquad` oder der Einheit 1 em.
Das ist die Breite von `\quad`: | |
Dabei ist "em" eine Größe, die von der jeweiligen Schrift abhängt.
in `\tenrm` (`\rm`) 10 pt
in `\tenbf` (`\bf`) 11,5 pt
in `\tensl` (`\sl`) 10 pt
in `\tenit` (`\it`) 10 pt
D `\def\quad{\hskip 1em\relax }`

∗ `\radical` TEX-Primitiv-Befehl zur Abarbeitung von `\sqrt`. Damit wird eine Abbildung in den Symbolfont mit den verschieden großen Wurzelzeichen vollzogen, um die Referenz auf die passenden Größen zu bilden.

`\raggedbottom` erlaubt, daß die Seiten beim Seitenumbruch in gewissem Umfang verschieden lang sein dürfen, um etwa Absätze nicht auseinander zu brechen. Dabei kommt die Fußzeile mit der Numerierung jedoch immer noch an die gleiche Stelle. Mittels `\normalbottom` wird dies wieder zurückgestellt. Dies ist auch die Voreinstellung.

```
D  \def\raggedbottom{\topskip 10\p@ plus60\p@
       \r@ggedbottomtrue }
```

`\raggedright` Textsatz: stellt Flatterrand beim Zeilenumbruch ein. 33
Jede Zeile darf bis zu '2em' kürzer als die Zeilenlänge sein. Der Leerraum zwischen den Wörtern kann nicht mehr wachsen.

```
D  \def\raggedright{\rightskip \z@ plus2em
                    \spaceskip .3333em
                    \xspaceskip .5em\relax }
```

∗ `\raise` kann im horizontalen Modus verwendet werden, um eine Box etwas anzuheben. 143
Beispiel: Was $^{\text{oben}}$ steht wurde mit
"`Was \raise3pt\hbox{oben} steht ...`" gesetzt
(siehe auch `\lower`, `\moveleft`, `\moveright`).

`\rangle` Mathematiksatz: rechte Klammer — ⟩ — 75
(wachsend in Kombination mit `\big..`, `\left`, `\right`)
(Gegenstück `\langle` — ⟨ —)
D `\def\rangle{\delimiter "526930B }`

`\rbrace` Mathematiksatz: rechte Klammer — } — 75
(wachsend in Kombination mit `\big..`, `\left`, `\right`)
(äquivalent mit `\}` — (Gegenstück `\lbrace`)
D `\def\rbrace{\delimiter "5267309 }`

`\rbrack` Mathematiksatz: rechte Klammer —] — 75
(wachsend in Kombination mit `\big..`, `\left`, `\right`)
(äquivalent mit `]` — (Gegenstück `\lbrack`)
D `\def\rbrack{]}`

\rceil — Mathematiksatz: rechte Klammer — ⌉ — 75
(wachsend in Kombination mit \big.., \left, \right)
(Gegenstück \lceil, siehe auch \lfloor, \rfloor)
D \def\rceil{\delimiter "5265307 }

\Re — Mathematiksatz: normales Zeichen — ℜ — 88
D \mathchardef\Re="23C

∗ \read — Eingabebefehl, um aus einer externen Datei einen Satz einzulesen. Dieser wird einem Befehlsnamen zugewiesen, der praktisch als Makroexpansion den Inhalt des Satzes besitzt. Eine Klammerstruktur in der Eingabe bewirkt, daß mehrere Sätze gelesen werden, um diese abzusättigen. 194, 195
Syntax: \read *n* to *name*
Beispiel:
```
\openin 7 = DATA
\read 7 to \meinedaten
```
Danach kann durch "\meinedaten" der Inhalt weiterverarbeitet werden
(siehe auch \closein).

∗ \relax — Dieser Befehl ist eine leere Anweisung: Er macht 'nichts'. In Makros findet er gelegentlich eine praktische Anwendung, indem er den vorangehenden Befehl beendet. Einige Befehle, zum Beispiel \hskip, besitzen ja eine variable Anzahl von Parametern. Durch ein nachgestelltes \relax wird ein solcher Befehl auf jeden Fall beendet.

\relbar — Mathematiksatz: Relation — − —
wird intern zum Aufbau von Pfeilen verwendet.
D \def\relbar{\mathrel {\smash -}}

\Relbar — Mathematiksatz: Relation — = —
(Wird intern zum Aufbau von Pfeilen verwendet;
D \def\Relbar{\mathrel =}

∗ \relpenalty — internes Register für die Minuspunkte, die beim Trennen einer mathematischen Formel im *textstyle* (nach einer Relation) aufgerechnet werden.
Vorbesetzt: \relpenalty=500

\removelastskip — 27
entfernt im vertikalen Modus den vorangehenden vertikalen *skip*, falls ein solcher vorhanden ist.
D
```
\def\removelastskip{\ifdim \lastskip =\z@
                    \else \vskip -\lastskip \fi }
```

\repeat — ist ein Schlüsselwort beim \loop-Befehl, und zwar die Endekennzeichnung. 126ff
Syntax: \loop *α* \if.. *β* \repeat
(siehe auch \loop)
D \let\repeat=\fi

`\rfloor` Mathematiksatz: rechte Klammer — ⌋ — 75
(wachsend in Kombination mit `\big..`, `\left`, `\right`)
(Gegenstück `\lfloor`, siehe auch `\lceil`, `\rceil`)
D `\def\rfloor{\delimiter "5263305 }`

`\rgroup` schließt eine Blockstruktur, die mit '`{`' oder `\bgroup` begonnen wurde. 75, *77*
D `\def\rgroup{\delimiter "500033B }`

`\rho` Mathematiksatz: griechischer Buchstabe — ρ — 66
(siehe auch `\varrho` — ϱ —)
D `\mathchardef\rho="11A`

`\rhook` Mathematiksatz: Hilfszeichen — ˒ — zur Konstruktion von `\hookleftarrow` — ↩ —
D `\mathchardef\rhook="312D`

∗ `\right` Mathematiksatz: schließt eine mit `\left` begonnene logische Klammerstruktur um eine Unterformel. Beiden Befehlen `\left` und `\right` folgt eine Angabe zu der Klammer, welche um diese Formel gesetzt werden soll. Diese wird dann in angepaßter Größe ausgegeben. 77, *79*, *84*
Beispiel:
`$$ \left[ x \over x-1 \right]$$`
ergibt

$$\left[\frac{x}{x-1}\right]$$

`\rightarrow` Mathematiksatz: Relation — → — 87
Der Befehl ist äquivalent mit `\to`
(siehe auch `\longrightarrow`).
D `\mathchardef\rightarrow="3221`

`\Rightarrow` Mathematiksatz: Relation — ⇒ — 87
(siehe auch `\Longrightarrow`)
D `\mathchardef\Rightarrow="3229`

`\rightarrowfill` 106
Textsatz: füllt eine Box soweit nötig mit einem Pfeil auf.
Beispiel:
`\hbox to 3cm{A\rightarrowfill B}`
ergibt

A⟶B

(siehe auch `\leftarrowfill`, `\hrulefill`)

D
```
\def\rightarrowfill{$\m@th \mathord -\mkern -6mu
     \cleaders \hbox {$\mkern -2mu
                    \mathord -\mkern -2mu$}\hfill
     \mkern -6mu\mathord \rightarrow $}
```

`\rightharpoondown` 87
Mathematiksatz: Relation — ⇁ —
D `\mathchardef\rightharpoondown="312B`

`\rightharpoonup` 87
Mathematiksatz: Relation — ⇀ —
D `\mathchardef\rightharpoonup="312A`

3 ∗ `\righthyphenmin`
ist ein neues internes Register, das angibt wie viele Zeichen beim Trennen mindestens abgetrennt werden müssen. Das Gegenstück ist `\lefthyphenmin`.
D `\lefthyphenmin=2 \righthyphenmin=3`

`\rightleftharpoons` 87
Mathematiksatz: Relation — ⇌ —
D
```
\def\rightleftharpoons
      {\mathrel {\mathpalette \rlh@ {}}}
```

`\rightline` Textsatz: Durch `\rightline{..text..}` wird der so angegebene Text rechtsbündig als einzelne Zeile gesetzt. Dieser Befehl sollte nur im vertikalen Modus, also nicht innerhalb eines Absatzes verwendet werden *18*
(siehe auch `\leftline`, `\centerline`).
D
```
\def\rightline#1{\line {\hss #1}}
\def\line{\hbox to\hsize }
```

∗ `\rightskip` Textsatz: gibt den Platz an, der rechts von jeder Zeile beim Absatzumbruch gelassen werden soll. Durch Setzung etwa von `\rightskip=1cm` bleibt rechts ein Rand von 1 cm. 36ff, 38
Vorbesetzt: `\rightskip=0pt`
(siehe auch `\leftskip`, `\narrower`)

`\rlap` *right lap* 142
Ausgabe von nach rechts überlappender Information
Beispiel: `ooo\rlap{//}uuu` liefert — ooo//uu —
(siehe auch `\llap` für o∅∅uuu)
D `\def\rlap#1{\hbox to\z@ {#1\hss }}`

`\rm` stellt die "roman" Schrift(-familie) (0) und gleichzeitg die roman Schrift in 10 Punkt ein. Dies ist die Standardschrift. 56
D `\def\rm{\fam \z@ \tenrm }`

`\rmoustache` Mathematiksatz: rechte Klammer — ⎱ — 75
(nur in Verbindung mit `\big..`, `\left`, `\right`)
(siehe auch `\lmoustache`)
D `\def\rmoustache{\delimiter "5000341 }`

∗ `\romannumeral` liefert kleine römische Zahlen (für Parameter ≥ 0).
`\romannumeral 1987` ergibt "mcmlxxxvii".
Für Großbuchstaben ist folgende Konstruktion nötig:
`\uppercase\expandafter{\romannumeral 1987}`
Dies ergibt "MCMLXXXVII"
(Für '1987' könnte zum Beispiel auch `\pageno` oder etwa `\count7` eingesetzt werden.)

`\root` Mathematiksatz: Makro zum Setzen von Wurzeln. 69
Beispiel:
`$$\root 3 \of {x+1}$$` ergibt

$$\sqrt[3]{x+1}$$

D
```
\def\root#1\of {\setbox \rootbox \hbox {$\m@th
      \scriptscriptstyle {#1}$} \mathpalette \r@@t }
```

`\rq` Ersatzbefehl *right quote* für ein Apostroph. 16
D `\def\rq{'}`

`\S` *section* liefert — § — 88
D `\def\S{\mathhexbox 278}`

`\sb` Mathematiksatz: Ersatzbefehl *subscript* für "_" 15
D `\let\sb=_`

`scaled` Schlüsselwort im `\font`-Befehl zur Angabe der Skalierung. Beispiel: *57*
`\font\bigbf=cmbx10 scaled \magstep2`

∗ `\scriptfont` Befehl zur Einstellung des *"scriptfont"* in einer Schriftfamilie; das ist die Schrift, die im Mathematiksatz für Exponenten und Indizes erster Stufe verwendet wird.
Zu den Standardeinstellungen gehört:
`\scriptfont0=\sevenrm` (roman)
`\scriptfont1=\seveni` (math italic)
Nach dem `\scriptfont` Befehl folgt die Nummer der Schriftfamilie (siehe auch `\fam`, `\newfam`).

∗ `\scriptscriptfont`
Befehl zur Einstellung des *"scriptscriptfont"* in einer Schriftfamilie, das ist die Schrift, die im Mathematiksatz für Exponenten und Indizes zweiter Stufe verwendet wird.
Beispiel einiger Standardeinstellungen:
`\scriptscriptfont0=\fiverm` (roman)
`\scriptscriptfont1=\fivei` (math italic)
(Dem `\scriptscriptfont`-Befehl folgt die Nummer der Schriftfamilie — siehe auch `\fam`, `\newfam`.)

∗ `\scriptscriptstyle` 68, *155*
Mathematiksatz: erzwingt die Darstellung einer Formel in der Form von Exponenten und Indizes zweiter Stufe.

∗ `\scriptspace` Mathematiksatz: Dies ist der zusätzliche Leerplatz nach einem Exponenten oder Index.
Vorbesetzt: `\scriptspace=0.5pt`

∗ `\scriptstyle` Mathematiksatz: erzwingt die Darstellung einer Formel in der Form von Exponenten und Indizes erster Stufe. 68, *155*

∗ `\scrollmode` setzt bei der Bearbeitung den *scrollmode*. Damit werden die Fehler auf dem Bildschirm noch protokolliert, aber es werden keine Korrekturanfragen mehr gestellt. Dieser Modus kann auch durch Eingabe von "`S`" bei einer Fehleranfrage gesetzt werden (siehe auch `\nonstopmode`, `\batchmode`). 164

`\searrow` Mathematiksatz: Relation — ↘ — *(south east arrow)* 87
D `\mathchardef\searrow="3226`

`\sec` Mathematiksatz: großer Operator — sec — 78
D `\def\sec{\mathop {\rm sec}\nolimits }`

∗ `\setbox` besetzt ein Box-Register (0 . . . 255) mit einer folgenden Box, wobei die Gruppenstrukturen, was den Inhalt betrifft, volle Gültigkeit besitzen. *51*, 132ff, *140*
Beispiel:
`\setbox0=\hbox{abcdef}`
Damit beinhaltet die Box 0 eine *horizontal box*. Diese kann mit `\box`, `\copy`, `\unhbox` und `\unhcopy` wieder ausgegeben werden. Eine `\vbox` kann mit `\box`, `\copy`, `\unvbox` und `\unvcopy` ausgegeben werden. Die Box-Register 0 bis 9 gelten als frei. Weitere benannte Registernummern lassen sich mit `\newbox` reservieren.

3 ∗ `\setlanguage` gefolgt von der Nummer einer einzustellenden Sprache, setzt ein sogenanntes *whatsit* Element mit der neuen Sprachnummer ab. Die Effekte sind allerdings sehr kompliziert, häufig wird dennoch nicht nach der so gesetzten Trenntabelle getrennt. Daher sollte statt dessen besser mit `\language` eine neue Sprache bzw. Trenntabelle eingestellt werden
(siehe besser `\language`).

`\setminus` Mathematiksatz: binärer Operator — \ — 86
Beispiel: `$A\setminus B$` ergibt $A \setminus B$.
(als normales Zeichen `\backslash`)
D `\mathchardef\setminus="226E`

`\settabs` Textsatz: definiert eine Folge von Tabulatoren. Dies geschieht entweder durch Angabe einer Spaltenanzahl oder durch Vorgabe einer Musterzeile. Beispiel: 91
`\settabs 5 \columns` (5 gleich breite Spalten)
`\settabs\+XXX&XXXXX&X\cr` (& ⇔ Tabulator)
(siehe auch `\cleartabs`, `\+`, `\cr`, `\tabalign`)

```
D \def\settabs
      {\setbox \tabs \null \futurelet \next \sett@b }
```

`\sevenbf` Anwahl der **boldface**-Schrift **in 7 Punkt.** 56
D `\font\sevenbf=cmbx7`

\seveni — Anwahl der 7 Punkt *math italic* Schrift. 58
D \font\seveni=cmmi7

\sevensy — Anwahl der 7 Punkt *mathematischen Symbole*. 58
D \font\sevensy=cmsy7

\sevenrm — Anwahl der 'roman'-Schrift in 7 Punkt. 56
D \font\sevenrm=cmr7

∗ \sfcode — *space factor* — Faktor, der jedem Zeichen eines Fonts zugeordnet ist und den Satz von nachfolgendem Leerraum regelt. Damit wird zum Beispiel nach Satzzeichen etwas mehr Leerplatz ausgegeben.

\sharp — Mathematiksatz: normales Zeichen — ♯ — 88
(siehe auch \flat, \natural)
D \mathchardef\sharp="15D

∗ \shipout — gibt den Inhalt eines Box-Registers während der *output-routine* in die Ausgabedatei *dvi-file* aus. 172ff, *175*

∗ \show — Protokollausgabe: gibt die Bedeutung der nachfolgenden Sequenz aus. Um den Inhalt zu erhalten, ist \showthe zu verwenden.

∗ \showbox — Protokollausgabe: \showbox wird zusammen mit der Nummer eines Box-Registers angegeben. Durch zum Beispiel \showbox0 wird der Inhalt des Registers 0 protokolliert.

∗ \showboxbreadth
steuert die Protokollausgabe von Boxen. Es gibt an, wie viele Elemente einer Box protokolliert werden sollen.
Voreingestellt: \showboxbreadth=5

∗ \showboxdepth — steuert die Protokollausgabe von Boxen. Es gibt die Verschachtelungstiefe an, für die noch eingeschachtelte Unterboxen protokolliert werden sollen.
Voreingestellt: \showboxdepth=3

\showhyphens — protokolliert die möglichen Trennstellen in einem Testtext. Beispiel: 52
\showhyphens{beispielsweise Urinstinkt} gibt

```
Underfull \hbox(badness 10000) detected at line 0
[] \tenrm bei-spiels-weise Ur-in-stinkt
```

D
```
\def\showhyphens#1{%
  \setbox0\vbox{\parfillskip\z@skip
    \hsize \maxdimen \tenrm \pretolerance \m@ne
    \tolerance \m@ne \hbadness 0\showboxdepth 0\ #1}}
```

∗ \showlists — Protokollausgabe: gibt die internen Arbeitslisten des TEX-Programms aus.

∗ \showthe — Protokollausgabe: gibt den *Inhalt* von Registern aus. (\show gibt die Bedeutung aus.)
(siehe auch \meaning)

`\sigma` Mathematiksatz: griechischer Buchstabe — σ — (siehe auch `\varsigma` — ς —) 66
D `\mathchardef\sigma="11B`

`\Sigma` Mathematiksatz: griechischer Buchstabe — Σ — 67
D `\mathchardef\Sigma="7006`

`\sim` Mathematiksatz: Relation — $\sim$ — 86
D `\mathchardef\sim="3218`

`\simeq` Mathematiksatz: Relation — $\simeq$ — 86
D `\mathchardef\simeq="3227`

`\sin` Mathematiksatz: großer Operator — sin — 78
D `\def\sin{\mathop {\rm sin}\nolimits }`

`\sinh` Mathematiksatz: großer Operator — sinh — 78
D `\def\sinh{\mathop {\rm sinh}\nolimits }`

`\skew` Mathematiksatz: Befehl zur Bildung mehrfacher Akzente. Syntax: `\skew{`*faktor*`}{`*oben*`}{`*unten*`}`
Der *faktor* gibt an, um wieviele `mu` *(math units)* der Akzent nach oben verschoben wird.
Beispiel:
`$\skew 6 \hat {\bar A}$` liefert $\skew{6}\hat{\bar{A}}$
D `\def\skew#1#2#3{{#2{#3\mkern #1mu}\mkern -#1mu}{}}`

∗ `\skewchar` Mathematiksatz: Durch `\skewchar` wird das schriftspezifische Referenzzeichen zur Positionierung von mathematischen Akzenten gebildet.
Definitionen in plain-TEX:
```
\skewchar\teni='177     \skewchar\seveni='177
\skewchar\fivei='177    \skewchar\tensy='60
\skewchar\sevensy='60   \skewchar\fivesy='60
```

∗ `\skip` referiert eines der 256 *skip*-Register (0 . . . 255), die mit Längenangaben besetzt werden können. Diese Längen dürfen auch Angaben für den möglichen Zuwachs- bzw. Schrumpfanteil besitzen. (Normale `\dimen`-Register dürfen nur reine Längenangaben zugewiesen bekommen.) Die Register 0 bis 9 gelten als frei. Durch etwa den Befehl `\newskip\meinskip` kann man sich ein weiteres freies Register zuweisen lassen und benutzen. 147
Beispiel:
```
\skip0=3cm plus 1cm minus 1cm
\hskip\skip0 oder \vskip\skip0
\newskip\meinskip  \meinskip=3cm plus 1cm
\vskip\meinskip
```

∗ **\skipdef** Dieser Befehl dient der Benennung eines *skip*-Registers. Beispiel:
`\skipdef\meinskip=7`
Intern wird dies in `\newskip` benutzt.
Besser ist es, `\newskip` zu verwenden.

\sl Anwahl der *slanted*-Schrift(-familie) (5). 56
D `\def\sl{\fam \slfam \tensl }`

\slash gibt ein "/" aus, wobei der Zeilenumbruch an dieser Stelle erlaubt ist. Beispiel: `Ein\slash Ausgabe`; dabei darf dieses Wort auch an "/" getrennt werden.
D `\def\slash{/\penalty \exhyphenpenalty }`

\slfam interne Nummer (5) der *slanted*-Schriftfamilie (siehe auch `\newfam`, `\sl`).
D
```
\newfam\slfam
\def\sl{\fam\slfam\tensl}
```

\smallbreak setzt ein `\smallskip` und gibt eine mögliche Seitenumbruchposition geringer Priorität an. 46
(siehe auch `\medbreak` und `\bigbreak`)
D
```
\def\smallbreak{\par
     \ifdim \lastskip <\smallskipamount
        \removelastskip \penalty -50\smallskip \fi }
```

\smallint Mathematiksatz: großer Operator — $\int$ —
(Allerdings wird gegenüber `\int` stets die kleinere Symbolgröße verwendet.)
D `\mathchardef\smallint="1273`

\smallskip Textsatz: setzt einen kleinen vertikalen Abstand vom Betrag `\smallskipamount`, gleichzeitig geht der Absatz zu Ende. Es gelten folgende Verhältnisse: 27
2 × `\smallskip` = `\medskip`
4 × `\smallskip` = `\bigskip` = 1 Leerzeile
D `\def\smallskip{\vskip \smallskipamount }`

\smallskipamount
ist der Umfang eines `\smallskip` —
D
```
\newskip\smallskipamount
\smallskipamount=3pt plus 1pt minus 1pt
```

\smash gibt in `\smash{..text..}` die angegebene Information aus, aber dabei wird so getan, als ob die Höhe und Tiefe (Unterlänge) Null seien 89, *104*
(siehe auch `\phantom`, `\vphantom`, `\hphantom`).
D
```
\def\smash
   {\relax \ifmmode
              \def \next {\mathpalette \mathsm@sh}%
           \else \let \next \makesm@sh \fi \next }
```

\smile Mathematiksatz: Relation — $\smile$ — 86
(Gegenstück: `\frown` — $\frown$ —)
D `\mathchardef\smile="315E`

`sp` interne Maßeinheit *scaled point*. 22
Mit dieser Genauigkeit wird intern gerechnet.
1 pt = 65536 sp

`\sp` Mathematiksatz: Ersatzbefehl *superscript* für "^" 15
`$x\sp 2$` ergibt — x^2 —
D `\let\sp=^`

`\space` gibt ein Leerzeichen aus. Mehrere Befehle `\space\space ...` erzeugen auch einen breiteren Leerraum in der Ausgabe.
D `\def\space{ }`

∗ `\spacefactor` Internes Register, das die Leerraumausgabe regelt. Es überschreibt den zeichenspezifischen `\sfcode`.

∗ `\spaceskip` Internes Register, das, wenn es von Null verschieden ist, den fontspezifischen Abstand zwischen zwei Wörtern überschreibt 29, *33*, *40*
(siehe auch `\xspaceskip`, `\fontdimen7 ...`).

`\spadesuit` Mathematiksatz: normales Zeichen — ♠ — 88
(siehe auch `\heartsuit`, `\diamondsuit`, `\clubsuit`)
D `\mathchardef\spadesuit="27F`

∗ `\span` Tabellensatz mittels `\halign`: Wird an Stelle von "`&`" in einer Eingabe `\span` verwendet, so werden zwei Tabellenelemente zusammengezogen und zusammenhängend gesetzt. Bei Angabe in der Musterzeile wird das folgende Element expandiert 107, *108*
(siehe auch `\multispan`).

∗ `\special` implementierungsabhängiger Befehl:
Durch `\special{ ... text ... }` wird zusätzlicher Text in die dvi-Datei geschrieben, der durch den Ausgabetreiber ausgewertet werden kann. Einige TEX-Implementierungen ermöglichen auf diese Weise grafische Ausgaben.

∗ `\splitbotmark` internes Register, welches den mittels `\mark` gesetzten Text ausgibt, der bei der letzten `\vsplit`-Operation gefunden wurde. Dies ist der letzte `\mark`-Text im abgesplitteten Text
(siehe auch `\mark`, `\botmark`, `\splitfirstmark`).

∗ `\splitfirstmark` internes Register, welches den mittels `\mark` gesetzten Text ausgibt, der bei der letzten `\vsplit`-Operation gefunden wurde. Dies ist der erste `\mark`-Text im abgesplitteten Text
(siehe auch `\mark`, `\botmark`, `\splitbotmark`).

∗ `\splitmaxdepth` ist die maximale Tiefe *depth* einer Box, die bei einer `\vsplit`-Operation abgespalten werden darf (siehe auch `\boxmaxdepth`).
Vorbesetzt: `\splitmaxdepth=\maxdimen` (beliebig)

∗ `\splittopskip` ist der Leerraum, der automatisch zu Beginn einer abgesplitteten Box gesetzt wird. 146
Vorbesetzt: `\splittopskip=10pt`
(Dies ist der gleiche Wert wie bei `\topskip`.)

`spread` Schlüsselwort bei der Angabe der Größe einer Box in einem `\halign`-, `\hbox`- oder `\vbox`-Befehl. Durch die folgende Dimension wird angegeben, um wieviel die folgende Box größer als ihre natürliche Weite sein soll. 101, 107, 134
Beispiel:
`\hbox spread 1cm{\hfill abc\hfill}`
erzeugt eine Box, die 1 cm größer als die durch ein einfaches `\hbox{abc}` erzeugte Box ist. Die Box sollte geeignet dynamischen Leerraum zum Auffüllen enthalten.

`\sqcap` Mathematiksatz: binärer Operator — ⊓ — 86
D `\mathchardef\sqcap="2275`

`\sqcup` Mathematiksatz: binärer Operator — ⊔ — 86
D `\mathchardef\sqcup="2274`

`\sqrt` Mathematiksatz: Setzen eines Wurzelzeichens. 69
Beispiel:
`$$ \sqrt { \sqrt{x+1} - 1} $$` erzeugt

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

D `\def\sqrt{\radical "270370 }`

`\sqsubseteq` Mathematiksatz: Relation — ⊑ — 86
D `\mathchardef\sqsubseteq="3276`

`\sqsupseteq` Mathematiksatz: Relation — ⊒ — 86
D `\mathchardef\sqsupseteq="3277`

`\ss` Textsatz : liefert — ß — 25
Ab TEX 3.0 kann das Zeichen ß mit einer geeigneten Eingabekodierung, wenn die betreffene Implementierung dies kann, auch direkt eingegeben werden. Im Notfall ist auch das folgende Rezept möglich:
`\catcode‘\ß=\active \letß=\ss`.
Die Platzkodierung `"19` bezieht sich auf die Computer Modern Schriften.
D `\chardef\ss="19`

`\star` Mathematiksatz: binärer Operator — ⋆ — 86
D `\mathchardef\star="213F`

* `\string` gibt den folgenden Befehl als String aus. Dabei wird das *escape*-Zeichen, normalerweise '\', durch das Zeichen mit dem Codewert von `\escapechar` dargestellt. Alle ausgegebenen Zeichen haben den `\catcode` 12 (sonstiges Zeichen). 126
Beispiel:
`\tt\string\par` erzeugt "`\par`"

`\strut` gibt eine leere Box aus, deren Höhe und Tiefe den Maßen einer normalen Zeile entsprechen. Damit kann ein gewisser Mindestzeilenabstand erzwungen werden, wenn dieser etwa mit `\offinterlineskip` abgeschaltet wurde. Im Tabellensatz mit senkrechten Strichen wird dieses Makro häufig benötigt. 102ff
Es wird die `\strutbox` ausgegeben mit einer Höhe von 8.5 pt und einer Tiefe von 3.5 pt
(siehe auch `\mathstrut` für den Mathematiksatz).

```
D \def\strut{\relax \ifmmode \copy \strutbox
                    \else \unhcopy \strutbox \fi }
```

`\strutbox` interne Nummer des Registers, das die für den `\strut`-Befehl nötige Box enthält. *51*

```
  \newbox\strutbox
D \setbox\strutbox=\hbox
        {\vrule height8.5pt depth3.5pt width\z@}
```

`\subset` Mathematiksatz: Relation — $\subset$ — 86

```
D \mathchardef\subset="321A
```

`\subseteq` Mathematiksatz: Relation — $\subseteq$ — 86

```
D \mathchardef\subseteq="3212
```

`\succ` Mathematiksatz: Relation — $\succ$ — 86

```
D \mathchardef\succ="321F
```

`\succeq` Mathematiksatz: Relation — $\succeq$ — 86

```
D \mathchardef\succeq="3217
```

`\sum` Mathematiksatz: großer Operator — $\sum$ — 73

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

`\sup` Mathematiksatz: großer Operator — sup — 78

```
D \def\sup{\mathop {\rm sup}}
```

`\supereject` erzwingt einen Aufruf der *output-routine*, indem extrem viele Pluspunkte (20000) für den Seitenumbruch erzeugt werden. 174
Dadurch werden durch die *output-routine* noch unverarbeitete Einfügungen ausgegeben.

```
D \def\supereject{\par \penalty -\@MM }
```

`\supset` Mathematiksatz: Relation — $\supset$ — 86

```
D \mathchardef\supset="321B
```

\supseteq Mathematiksatz: Relation — $\supseteq$ — 86
D \mathchardef\supseteq="3213

\surd Mathematiksatz: normales Zeichen — $\surd$ — 88
D \def\surd{{\mathchar "1270}}

\swarrow Mathematiksatz: Relation — $\swarrow$ — 87
(south west arrow)
D \mathchardef\swarrow="322E

\t Textsatz: *tie-after*-Akzent \t oo gibt — o͡o — 25
D \def\t#1{{\edef \next {\the \font }%
\the \textfont 1\accent "7F\next #1}}

\tabalign ist ein internes plain-TEX-Makro zur Tabellenerzeugung. Es wirkt wie ein '\+', darf aber auch "innen", das heißt innerhalb von Makros, angewendet werden, im Gegensatz zu "\+", das als "\outer" deklariert ist.
D \def\tabalign{\us@true \m@ketabbox }

\tabs ist eine interne plain-TEX-Hilfsbox zur Abwicklung der Befehle \settabs, \+, \cleartabs.
D \newbox\tabs % wird zu \chardef\tabs="C

\tabsdone ist eine interne plain-TEX-Hilfsbox zur Abwicklung der Befehle \settabs, \+, \cleartabs.
D \newbox\tabsdone % wird zu \chardef\tabsdone="E

\tabsyet ist interne plain-TEX-Hilfsbox zur Abwicklung der Befehle \settabs, \+, \cleartabs.
D \newbox\tabsyet % wird zu \chardef\tabsyet="D

∗ \tabskip ist das Standardregister für den zusätzlichen Leerraum, der beim \halign-Befehl vor, zwischen und hinter die Tabellenspalten gesetzt wird. *100*
Der zu Beginn von \halign gültige \tabskip wird *vor* die erste Spalte gesetzt; der beim Ende der Musterzeile, also beim \cr gültige \tabskip wird *nach* der letzten Spalte gesetzt.
Zwischen den Spalten wird der jeweils zu diesem Zeitpunkt gültige \tabskip gesetzt.

\tan Mathematiksatz: großer Operator — tan — 78
D \def\tan{\mathop {\rm tan}\nolimits }

\tanh Mathematiksatz: großer Operator — tanh — 78
D \def\tanh{\mathop {\rm tanh}\nolimits }

\tau Mathematiksatz: griechischer Buchstabe — τ — 66
D \mathchardef\tau="11C

\tenbf Name und Anwahl der **boldface**-Schrift in 10 Punkt. 56, 58
D \font\tenbf=cmbx10

\tenex — ist der definierte Name des Fonts für große Symbole im Mathematiksatz (10 Punkt Größe). 58
D \font\tenex=cmex10

\teni — Name und Anwahl der "*math italic*" Schrift. 58
D \font\teni=cmmi10

\tenit — Name und Anwahl der *"italic"* Schrift. 56, 58
D \font\tenit=cmti10

\tenrm — Name und Anwahl der normalen "roman" Schrift in 10 Punkt. *47*, 56, 58
D \font\tenrm=cmr10

\tensl — Name und Anwahl der normalen *"slanted"* Schrift 10 Punkt. 56, 58
D \font\tensl=cmsl10

\tensy — ist der definierte Name des Fonts für normale Symbole im Mathematiksatz (10 Punkt Größe). 58
D \font\tensy=cmsy10

\tentt — Name und Anwahl der `typewriter`-Schrift in 10 Punkt. 56, 58
D \font\tentt=cmtt10

\TeX — Dieser Befehl liefert — TEX —
D
```
\def\TeX
   {T\kern -.1667em\lower .5ex\hbox{E}\kern -.125emX}
```

* \textfont — setzt den *textfont* zu einer Schriftfamilie.
So ist standardmäßig definiert:
```
\textfont0=\tenrm
\textfont1=\teni
\textfont\bffam=\tenbf
```
(siehe auch \fam, \newfam, \scriptfont)

\textindent — ist ein plain-TEX Hilfsmakro zu \item und \itemitem.
D
```
\def\textindent#1{\indent
                  \llap {#1\enspace}\ignorespaces}
```

* \textstyle — Mathematiksatz: erzwingt die Ausgabe einer Formel im *text-style*, das heißt, die Ausgabe ist genauso als wäre die Formel normal zwischen $...$ gesetzt. 70, *155*
(siehe auch \scriptstyle, \displaystyle und \scriptscriptstyle)

* \the — Allgemeiner Ausgabebefehl für eine interne Größe oder Register des TEX-Programms. Dies können *token*-Register zum Beispiel \the\headline, \the\everypar oder \the\skip5 usw. sein. *173*
(Der Befehl \showthe protokolliert, wie die Ausgabe mittels \the sein würde.)

\theta — Mathematiksatz: griechischer Buchstabe — θ — 66
(siehe auch \vartheta — ϑ —)
D \mathchardef\theta="112

	`\Theta`	Mathematiksatz: griechischer Buchstabe — Θ — D `\mathchardef\Theta="7002`	67
∗	`\thickmuskip`	Mathematiksatz: großer Leerraum in Einheiten von `mu` = *math units* Vorbesetzt: `\thickmuskip=5mu plus 5mu` (siehe auch `\thinmuskip`, `\medmuskip`)	78, 151
∗	`\thinmuskip`	Mathematiksatz: kleiner Leerraum in Einheiten von `mu` = *math units* Vorbesetzt: `\thinmuskip=3mu` (siehe auch `\thickmuskip`, `\medmuskip`)	78, 151
	`\thinspace`	Textsatz: kleiner Leerraum 1/6 *em* (siehe auch `\negthinspace`) D `\def\thinspace{\kern .16667em }`	29
	`\tilde`	Mathematiksatz: Akzent `$\tilde x$` liefert — $\tilde{x}$ — D `\def\tilde{\mathaccent "707E }`	69
∗	`\time`	enthält die Uhrzeit (in Minuten seit Mitternacht) zum Zeitpunkt des Programmstarts. Dies wird mit `\the\time` ausgegeben (siehe auch `\day`, `\month`, `\year`).	
	`\times`	Mathematiksatz: binärer Operator — × — D `\mathchardef\times="2202`	86
	`to`	Schlüsselwort in `\hbox`, `\vbox`, `\vtop` Befehlen Beispiel: `\hbox to 3cm{...}`	134
	`\to`	Mathematiksatz: Relation — → — (äquivalent ist `\rightarrow`) D `\mathchardef\to="3221`	87
∗	`\toks`	referiert eines von 256 *token*-Registern. Die am häufigsten verwendeten Register sind `\headline` und `\footline`. *token*-Register werden mittels Zuweisung besetzt, sie erhalten dann — ähnlich wie bei einer Makrodefinition — eine Folge von Befehlen, die durch `\the...` dann ausgegeben und interpretiert wird. Beipspiel: `\toks0={\bigskip\hrule\bigskip}` Im Normalfall erzeugt man mit `\newtoks` ein freies benanntes neues Register. `\newtoks\meintok` `\meintok={\bigskip\hrule\bigskip}` `\the\meintok` — ist dann die Ausgabe.	148
∗	`\toksdef`	benennt ein *token*-Register, dessen Nummer bekannt ist. Nach `\toksdef\mytoks=0` kann das *token*-Register 0 unter dem Namen `\mytoks` benutzt werden. Dieser Befehl wird auch bei `\newtoks` zur Benennung verwendet.	

∗ `\tolerance` Bewertungsgrenzwert für noch zu akzeptierende schlecht umbrochene Zeilen, und zwar mit erlaubten Trennungen. Wird `\tolerance` vergrößert, so kommt es auch zu mehr Leerraum zwischen den einzelnen Wörtern. Voreinstellung: `\tolerance=200` (siehe auch `\pretolerance`, `\emergencystretch`) 42

`\top` Mathematiksatz: binärer Operator — ⊤ — (siehe auch `\bot` für — ⊥ —) 88
D `\mathchardef\top="23E`

3 `\topglue` wird aufgerufen mit einer nachfolgenden Längenangabe, beispielsweise `\topglue 3cm`. Es ist nur interessant, wenn es am Anfang einer Seite gesetzt wird. Dort entfernt es den `\topskip` und erzeugt auf jeden Fall Leerraum der angegebenen Größe. Im Gegensatz zum Register `\topskip`, das den Abstand bis zur Grundline der ersten Zeile/Box auf der Seite bestimmt, wird hier der Leerraum bis zur Oberkante der ersten Box berechnet. 28
D
```
\def\topglue{\nointerlineskip
             \vglue-\topskip\vglue}
```

`\topins` ist ein Einfügungsregister für `\topinsert`-Befehle. 173
D `\newinsert\topins % wird zu: \chardef\topins="FD`

`\topinsert` Befehl zur Einfügung von Information an den Anfang der aktuellen Seite; ist nicht mehr genügend Platz vorhanden, so wird die angegebene Information an den Anfang der folgenden Seite gesetzt. 49
Syntax: `\topinsert` *vertikales Material* `\endinsert`
(siehe auch `\midinsert` und `\pageinsert`)
D `\def\topinsert{\@midfalse \p@gefalse \@ins }`

∗ `\topmark` liefert den mittels `\mark` gesetzten Markierungstext, und zwar den, der zu Beginn der aktuellen Seite gültig ist. Das ist derjenige, der als letzter zum Ende der Vorgängerseite galt 178
(siehe auch `\botmark`, `\mark`, `\firstmark`).

∗ `\topskip` Ist der Mindestabstand, den die Grundlinie der ersten Zeile (Box) einer jeden Seite vom Seitenanfang haben soll. 28, 38
Vorbesetzt: `\topskip=10pt`
(siehe auch `\splittopskip`, `\topglue`)

`\tracingall` aktiviert alle Testparameter. Anschließend erhält man das Maximum an Testausgaben! 170

```
\def\tracingall
    {\tracingonline \@ne   \tracingcommands \tw@
     \tracingstats \tw@    \tracingpages \@ne
     \tracingoutput \@ne   \tracinglostchars \@ne
D    \tracingmacros \tw@   \tracingparagraphs \@ne
     \tracingrestores \@ne
     \showboxbreadth \maxdimen
     \showboxdepth \maxdimen
     \errorstopmode }
```

* `\tracingcommands` 170

Parameter, der angibt, wie ausführlich die eingegebenen Kommandos protokolliert werden sollen.
Voreingestellt: `\tracingcommands=0`
Zusätzlich erhält man bei `\tracingcommands=`

- `1` jedes ausgeführte Kommando,
- `2` auch nicht ausgeführte Kommandos aus `\else`-Zweigen in Abfragen protokolliert.

* `\tracinglostchars`

Protokollparameter: aktiviert, falls größer Null, die Protokollierung von Zeichen, die zwar referiert wurden, aber in dem betreffenden Font nicht vorhanden sind.
Voreingestellt: `\tracinglostchars=1`

* `\tracingmacros` Protokollparameter: aktiviert, wenn er auf einen Wert größer Null gesetzt wird, die Protokollierung der Makroparameter. 170
Vorbesetzt: `\tracingmacros=0`

* `\tracingonline` gibt an, ob die Testausgaben der `\tracing..`-Befehle auch auf dem Terminal erscheinen sollen. Dies geschieht für `\tracingonline` > 0 170
Vorbesetzt: `\tracingonline=0`

* `\tracingoutput` Für Werte größer als Null wird der Inhalt der mittels `\shipout` erzeugten Seite protokolliert.
Vorbesetzt: `\tracingoutput=0`

* `\tracingpages` Für Werte größer als Null wird die Erstellung des Seitenumbruchs protokolliert. Dabei wird nach jeder Box, die zu der aktuellen Ausgabeseite hinzugefügt wird, folgendes ausgegeben:

`\pagetotal`	(t) bisheriger Informationsumfang
`\pagegoal`	(g) angestrebte Seitengröße
badness	(b) aktuelle Gewichtung
penalty	(p) Bewertung einer Umbruchstelle
costs	(c) Gesamtbewertung ($c = p + b$)

* `\tracingparagraphs`

Nach `\tracingparagraphs=1` wird der Zeilenumbruch protokolliert.

∗ `\tracingrestores`
Für `\tracingrestores=1` werden die Umsetzungen beziehungsweise Zurücksetzungen von veränderten Werten beim Schließen einer Gruppe protokolliert.

∗ `\tracingstats` für `\tracingstats=1` wird am Ende des Programmlaufs eine Statistik des benutzten Programmspeichers ausgegeben, für `\tracingstats=2` geschieht dies auch am Ende jeder Seite. 170
Vorbesetzt: `\tracingstats=0`

`\triangle` Mathematiksatz: normales Zeichen — △ — 88
D `\mathchardef\triangle="234`

`\triangleleft` Mathematiksatz: binärer Operator — ◁ — 86
D `\mathchardef\triangleleft="212F`

`\triangleright` Mathematiksatz: binärer Operator — ▷ — 86
D `\mathchardef\triangleright="212E`

`true` Schlüsselwort bei Längenangaben: Wird es verwendet, so wird die betreffende Länge *nicht* aufgrund einer vorliegenden `\magnification`-Angabe skaliert. *23*, *37*, *50*
Beispiel: `\vskip 10 true cm` für eine Abbildung

`\tt` Anwahl der `typewriter`-Schrift(-familie). 56
D `\def\tt{\fam \ttfam \tentt }`

`\ttfam` enthält die interne Nummer (7) der `typewriter`-Schriftfamilie (siehe auch `\newfam`, `\tt`).
D `\newfam\ttfam`
`\def\tt{\fam\ttfam\tentt}`

`\ttraggedright` ist eine Sonderform zur Einstellung des Flatterrandes (siehe auch `\raggedright`) bei der `typewriter`-Schrift. 33
D `\def\ttraggedright{\tt \rightskip \z@ plus2em\relax }`

`\u` Textsatz: Akzent `\u o` liefert — ŏ — 25
D `\def\u#1{{\accent 21 #1}}`

∗ `\uccode` *upper case code* Zeichen, das jedem der möglichen 256 Zeichen zugeordnet ist. Es bestimmt, in welches Symbol das betreffende Zeichen beim `\uppercase`-Befehl umgewandelt wird. So ist `\uccode‘\a=‘\A` definiert (siehe auch `\lccode` für `\lowercase`).

∗ `\uchyph` ist ein interner Parameter im Textsatz:
Dieser gibt für `\uchyph` > 0 an, daß auch Wörter, die mit einem Großbuchstaben beginnen, getrennt werden dürfen.
Voreingestellt: `\uchyph=1`

`\underbar` Textsatz: unterstreicht die Information auf Parameterposition, aber man beachte den Effekt auf Unterlängen.
Beispiel:
`\underbar{ganz geheim}` liefert — ganz geheim —
(Die Unterlängen werden durchgestrichen!)

```
D \def\underbar#1{$\setbox \z@ \hbox {#1}%
        \dp \z@ \z@
        \m@th \underline {\box \z@ }$}
```

`\underbrace` Mathematiksatz: setzt eine Klammer unter die Information. Beispiel: 83
`$$\underbrace{x+y+z}_{>0}$$` erzeugt

$$\underbrace{x+y+z}_{>0}$$

(siehe auch `\overbrace`)

```
D \def\underbrace#1{\mathop
     {\vtop {\ialign {##\crcr
        $\hfil \displaystyle {#1}\hfil $\crcr
        \noalign{\kern 3\p@ \nointerlineskip }
        \upbracefill \crcr
        \noalign{\kern 3\p@ }}}}\limits }
```

∗ `\underline` Mathematiksatz: unterstreicht die Information auf Parameterposition. 69
Beispiel:
`$\underline{A+B}$` liefert $\underline{A+B}$
(siehe auch `\overline`)

∗ `\unhbox` gibt eine `\hbox` aus, wobei die äußere (`\hbox`-Struktur) wieder entfernt wird. Die Box wird sozusagen einmal entklammert. 133
Beispiel:

```
\setbox1=\hbox{AB}
\setbox2=\hbox{\unhbox1 C}
```

Dies ist äquivalent zu

```
\setbox2=\hbox{ABC}
```

Dagegen ist

```
\setbox3=\hbox{\box1 C}
```

äquivalent zu

```
\setbox3=\hbox{\hbox{AB}C}
```

Die bei `\unhbox` referierte Box ist anschließend leer.

∗ `\unhcopy` wirkt wie `\unhbox`, jedoch bleibt die referierte Box erhalten, sie wird nur kopiert. 133

∗ `\unkern` entfernt einen vorangehenden `\kern` in der Arbeitsliste, falls ein solcher vorhanden ist.

∗ `\unpenalty` entfernt einen vorangehenden `\penalty` in der Arbeitsliste, falls ein solcher vorhanden ist.

∗ `\unskip` entfernt einen vorangehenden `\hskip` oder `\vskip` aus der Arbeitsliste, falls ein solcher vorhanden ist. 106

∗ `\unvbox` gibt eine `\vbox` unter Entfernung der äußeren `\vbox`-Struktur aus. Die referierte Box ist anschließend leer. 133
Beispiel:
```
\setbox1=\vbox{\hbox{AB}}
\setbox2=\vbox{\unvbox1\hbox{C}}
```
Dies ist äquivalent zu
```
\setbox2=\vbox{\hbox{AB}\hbox{C}}
```
Dagegen ist
```
\setbox3=\vbox{\box1\hbox{C}}
```
äquivalent zu
```
\setbox3=\vbox{\vbox{\hbox{AB}}\hbox{C}}
```

∗ `\unvcopy` gibt wie `\unvbox` die Box aus, jedoch bleibt ihr alter Inhalt noch erhalten. 133

`\uparrow` Mathematiksatz: Relation — ↑ — 75, 87
D `\def\uparrow{\delimiter "3222378 }`

`\Uparrow` Mathematiksatz: Relation — ⇑ — 75, 87
D `\def\Uparrow{\delimiter "322A37E }`

`\upbracefill` Text-, Tabellensatz: füllt eine Box in der nötigen Breite mit einer Klammer auf: 106
`\hbox to 3cm{\upbracefill}` ergibt
(siehe auch `\downbracefill`)
D
```
\def\upbracefill{$\m@th \bracelu
    \leaders\vrule\hfill
    \bracerd \braceld
    \leaders\vrule\hfill \braceru $}
```

`\updownarrow` Mathematiksatz: Relation — ↕ — 75, 87
D `\def\updownarrow{\delimiter "326C33F }`

`\Updownarrow` Mathematiksatz: Relation — ⇕ — 75, 87
D `\def\Updownarrow{\delimiter "326D377 }`

`\uplus` Mathematiksatz: binärer Operator — ⊎ — 86
D `\mathchardef\uplus="225D`

∗ `\uppercase` bildet Großbuchstaben aus der Eingabe.
Durch `\uppercase{aBc}` wird die Eingabe zu `ABC`. Makros, die in der Eingabe enthalten sind, werden *nicht* expandiert. Dies wird durch folgende Konstruktion erreicht:
```
\edef\next{...information...}
\uppercase\expandafter{\next}
```
(siehe auch `\lowercase`)

`\upsilon` Mathematiksatz: griechischer Buchstabe — υ — 66
D `\mathchardef\upsilon="11D`

`\Upsilon` Mathematiksatz: griechischer Buchstabe — Υ — 67
D `\mathchardef\Upsilon="7007`

`\v` Textsatz: Akzent `\v o` liefert — ŏ — 25
D `\def\v#1{{\accent 20 #1}}`

∗ `\vadjust` Textsatz: Durch `\vadjust{`*vertikale Information*`}` wird im normalen Absatzumbruch die bei `\vadjust` angegebene Information nach der aktuell umbrochenen Zeile eingefügt. Das Beispiel `\vadjust{\smallskip}` führt dazu, daß nach der gesetzten Zeile, in der dies steht, ein `\smallskip` ausgeführt wird. 28, 144

∗ `\valign` Textsatz: Dieser Befehl entspricht dem `\halign`-Befehl, jedoch sind Zeilen und Spalten in ihrer Bedeutung vertauscht!

`\varepsilon` Mathematiksatz: griechischer Buchstabe — ε — 66
(siehe auch `\epsilon` — ϵ —)
D `\mathchardef\varepsilon="122`

`\varphi` Mathematiksatz: griechischer Buchstabe — φ — 66
(siehe auch `\phi` — ϕ —)
D `\mathchardef\varphi="127`

`\varpi` Mathematiksatz: griechischer Buchstabe — ϖ — 66
(siehe auch `\pi` — π —)
D `\mathchardef\varpi="124`

`\varrho` Mathematiksatz: griechischer Buchstabe — ϱ — 66
(siehe auch `\rho` — ρ —)
D `\mathchardef\varrho="125`

`\varsigma` Mathematiksatz: griechischer Buchstabe — ς — 66
(siehe auch `\sigma` — σ —)
D `\mathchardef\varsigma="126`

`\vartheta` Mathematiksatz: griechischer Buchstabe — ϑ — 66
(siehe auch `\theta` — θ —)
D `\mathchardef\vartheta="123`

∗ `\vbadness` ist der Grenzwert, ab dem eine *schlechte* `\vbox` protokolliert wird.
Vorbesetzt: `\vbadness=1000`
(siehe auch `\hbadness`, `\vfuzz`, `\hfuzz`)

∗ `\vbox` Durch `\vbox{` *information* `}` wird eine *vertical box* gebildet, in der Information untereinander gesetzt wird. Einige Befehle oder Zeichen vollziehen jedoch auch automatisch einen Moduswechsel, so daß man sich im *horizontal mode*, das heißt im Absatzumbruch, befindet. Direkt untereinander gesetzt werden in einer `\vbox` wieder: eine andere Box (`\hbox`, `\vbox`, `\vtop`, `\vcenter`), vertikaler Leerraum (`\vskip`..., `\smallskip`...) und Striche mittels `\hrule`. Einfacher Text wie "`\vbox{Hier geht es ...`" führt zum Umbruch entsprechend der aktuellen `\hsize`. Die Grundlinie einer `\vbox` bestimmt sich aus der Grundlinie der untersten Box/Zeile; genauso die Unterlänge
(siehe auch `\vtop` und `\vcenter`). — *48*, *94*, 130ff, *135*

∗ `\vcenter` Mathematiksatz: zentriert die folgende Box vertikal, wobei noch eine Korrektur um den Wert der mathematischen Symmetrieachse (`\fontdimen22\tensy`) erfolgt. — 131, *175*

`\vdash` Mathematiksatz: Relation — $\vdash$ —
(siehe auch `\dashv` — $\dashv$ —)
D `\mathchardef\vdash="3260` — 86

`\vdots` Mathematiksatz: *vertical dots* liefert — $\vdots$ —
(siehe auch `\ldots`, `\ddots`, `\cdots` — $\ldots$ $\ddots$ $\cdots$ —
D
```
\def\vdots{\vbox {\baselineskip 4\p@
        \lineskiplimit \z@
        \kern 6\p@ \hbox {.}\hbox {.}\hbox {.}}}
```
— 80

`\vec` Mathematiksatz: mathematischer Akzent.
`$\vec x$` liefert — $\vec{x}$ —
D `\def\vec{\mathaccent "017E }` — 70

`\vee` Mathematiksatz: binärer Operator — $\vee$ —
(äquivalent zu `\lor`)
D `\mathchardef\vee="225F` — 86

`\vert` Mathematiksatz: normales Zeichen — | —
(äquivalent mit "`|`") — 75, *80*, 88
D `\def\vert{\delimiter "26A30C }`

`\Vert` Mathematiksatz: normales Zeichen — ‖ —
(äqivalent mit "`\|`"; dagegen als Relation `\parallel`) — 75, *77*, 88
D `\def\Vert{\delimiter "26B30D }`

∗ `\vfil` erzeugt wachsenden vertikalen Leerraum ersten Grades. — 142

∗ `\vfill` erzeugt wachsenden vertikalen Leerraum zweiten Grades. — 45, 142

∗ `\vfilneg` entfernt ein eventuell vorangehendes `\vfil`. — 142

`\vfootnote` Hilfsmakro von plain-TEX zur Abwicklung der Fußnoten. Es genügt der gleichen Syntax wie `\footnote`. Der Unterschied ist, daß das Markierungszeichen im Text *nicht* gesetzt wird.

```
\def\vfootnote#1{\insert \footins \bgroup
    \interlinepenalty \interfootnotelinepenalty
    \splittopskip \ht \strutbox
    \splitmaxdepth \dp \strutbox
    \floatingpenalty \@MM
D   \leftskip \z@skip
    \rightskip \z@skip
    \spaceskip \z@skip
    \xspaceskip \z@skip
    \textindent {#1}\footstrut
    \futurelet \next \fo@t }
```

∗ `\vfuzz` Grenzwert, ab dem die Überfüllung einer `\vbox` protokolliert wird. 54
Vorbesetzt: `\vfuzz=0pt`
(siehe auch `\hfuzz`)

`\vglue` erzeugt vertikalen Leerraum, der auch beim Umbruch nicht entfernt werden kann, indem `\nobreak` und leere `\hrule`s eingefügt werden. 28
Beispiel: `\vglue 3cm plus 1 cm`
(`\hglue` ist auch vorhanden.)

```
\def\vglue{\afterassignment \vgl@ \skip@ =}
D \def\vgl@{\par \dimen@\prevdepth \hrule height\z@
            \nobreak\vskip\skip@ \prevdepth\dimen@}
```

∗ `\voffset` ist der vertikale Versatz, mit dem die gesamte Ausgabe bei der Druckausgabe relativ zum Papier nach unten verschoben werden soll. Vorsicht! 37, *37*, 38
Vorbesetzt: `\voffset=0pt`

`\vphantom` Durch `\vphantom{...information...}` wird nur die aktuelle Zeile oder Box auf mindestens die Höhe und Tiefe gebracht, wie sie die Information des Parameters hätte. Die Breite der ausgegebenen Information ist Null (siehe auch `\phantom`, `\hphantom`, `\smash`). 89

```
D \def\vphantom{\v@true \h@false \ph@nt }
```

∗ `\vrule` zieht eine senkrechte Linie. Dieser Befehl ist nur im horizontalen Modus anwendbar. Im vertikalen Modus bewirkt er den Übergang in den horizontalen Modus. 53, 102ff, *136*
Syntax:

`\vrule width` *dim* `height` *dim* `depth` *dim*

Die Parameter sind optional, sie geben Breite und Höhe und Tiefe des zu erzeugenden Strichs an. `\vrule` ohne Parameter erzeugt einen Strich mit der Höhe der umgebenden Box (Zeile) und der Breite von 0.4 pt.

*	`\vsize`	ist die Höhe des Satzspiegels. Nach dieser Größe erfolgt der Seitenumbruch. Sie kann lokal umgestellt werden. Voreingestellt: `\vsize=8.9 true in` (siehe auch `\hsize` für die Satzbreite/Zeilenlänge im Umbruch)	21, 37, 38
*	`\vskip`	gibt vertikalen Leerraum aus. In einem Absatz gegeben, wird dieser beendet. Leerraum am Seitenanfang wird entfernt. Beispiel: `\vskip 1 cm plus 0.5cm` (siehe auch `\vglue`, `\topglue`)	27
*	`\vsplit`	Mit dieser Operation kann aus einer `\vbox` gemäß einer Längenangabe das obere Stück an einer möglichst guten Umbruchstelle abgespalten werden. Beispiel: `\setbox0=\vsplit 1 to 5 cm` Damit wird von dem Inhalt von Box 1 der erste Teil mit einer möglichst passenden Länge von 5 cm abgespalten. Die Box 1 enthält hinterher noch den Rest. `\splittopskip` ist der *skip*, der in der Zielbox oben automatisch eingefügt wird, `\splitmaxdepth` die maximale Unterlänge der Zielbox. Vorbesetzt: `\splitmaxdepth=\maxdimen`	*128*, 145ff
*	`\vss`	setzt im vertikalen Modus oder in einer `\vbox` beliebig wachsenden oder schrumpfenden Leerraum. Ist `\vss` in einer `\vbox` enthalten, kann nie eine Fehlermeldung wegen einer Über- oder Unterfüllung entstehen.	142
*	`\vtop`	bildet wie `\vbox` eine vertikale Box: Die Grundlinie wird nicht nach der untersten Box/Zeile, sondern nach der obersten bestimmt. Eine solche Box hat dann also eine große Unterlänge, nämlich aus dem Rest aller folgenden Unterboxen.	131, *146*
*	`\wd`	Damit kann die Breite eines der 256 Box-Register referiert werden. Beispiel: `\setbox0=\hbox{Anton}` Danach enthält `\wd0` die Breite des Textes "`Anton`" (siehe auch `\dp` und `\ht`).	134
	`\wedge`	Mathematiksatz: binärer Operator — $\wedge$ — (äquivalent ist `\land`) D `\mathchardef\wedge="225E`	86
	`\widehat`	Mathematiksatz: breiter mathematischer Akzent. `$\widehat{xyz}$` liefert — $\widehat{xyz}$ — D `\def\widehat{\mathaccent "0362 }`	70

`\widetilde` Mathematiksatz: breiter mathematischer Akzent. 70
`$\widetilde{xyz}$` liefert — $\widetilde{xyz}$ —
D `\def\widetilde{\mathaccent "0365 }`

∗ `\widowpenalty` Minuspunkte, die beim Seitenumbruch aufgerechnet werden, falls die letzte Zeile gerade noch auf die nächste Seite umbrochen wird.
Vorbesetzt: `\widowpenalty=150`
(siehe auch `\displaywidowpenalty`)

`width` Schlüsselwort bei Längenangaben zu `\hrule`, `\vrule`. 53
Beispiel: `\hrule width 10 cm`

`\wlog` Die mittels `\wlog{...text...}` angegebene Information wird (nur) in die Protokolldatei geschrieben.
D `\def\wlog{\immediate \write \m@ne }`

`\wp` Mathematiksatz: normales Zeichen — $\wp$ — 88
D `\mathchardef\wp="17D`

`\wr` Mathematiksatz: binärer Operator — $\wr$ — 86
D `\mathchardef\wr="226F`

∗ `\write` Mittels `\write` wird eine Textfolge in eine externe Ausgabedatei geschrieben werden.
Syntax: `\write` *n* `{...daten...}`
n ist dabei die Nummer eines Ausgabestroms oder die mittels `\newwrite` benannte Nummer. Die Ausgabe erfolgt erst während der *output routine*, es sei denn ein `\immediate` ist dem `\write` Befehl vorangestellt.
Beispiel: 194, *196*, *198*

`\openout 7=AUSGABE`	Öffnung
`\write 7{...daten...}`	Schreiben
`\closeout7`	Schließen

(siehe auch `\read`, `\openin`, `\closein`, `\immediate`)

∗ `\xdef` Befehl zur Definition von Makros: Dieser ist äquivalent zu `\global\edef`. Damit wird global ein Makro definiert, wobei in dem Definitionsteil vorkommende weitere Aufrufe schon zur Definitionszeit expandiert werden 114, 198
(siehe auch `\edef`, `\def`, `\global`, `\noexpand`).

`\xi` Mathematiksatz: griechischer Buchstabe — ξ — 66
D `\mathchardef\xi="118`

`\Xi` Mathematiksatz: griechischer Buchstabe — Ξ — 67
D `\mathchardef\Xi="7004`

∗ `\xleaders` wiederholt die folgende Box oder `\hrule` bis zur Breite, die durch das folgende `\hskip` bzw. `\hfill` angegeben wird.
Beispiel:
```
\def\xleaderfill
  {\xleaders\hbox{\tt....+....*}\hfill}
\line{\xleaderfill}
```
liefert
```
  ....+....*  ....+....*  ....+....*  ....+....*
```
Dabei werden die entstehenden Boxen so ausgerichtet, daß überflüssiger Leerraum gleichmäßig zwischen den einzelnen Boxen aufgeteilt wird
(siehe auch `\cleaders`, `\leaders`).

∗ `\xspaceskip` überschreibt — falls von Null verschieden — den schriftspezifischen Parameter für den Leerraum am Satzende. Damit wird `\fontdimen7<`*font*`>` wirkungslos. 29, *33*

∗ `\year` Register mit der Jahreszahl des Datums, an dem das Programm gestartet wurde. Es wird durch `\the\year` ausgegeben (siehe auch `\day`, `\month`, `\time`).

`\zeta` Mathematiksatz: griechischer Buchstabe — ζ — 66
D `\mathchardef\zeta="110`

14.2 Versteckte plain-TEX-Befehle

Die hier aufgeführten Befehle sind aus Gründen der Optimierung des Speicherplatzes oder des Laufzeitverhaltens entstanden. So ist es beispielsweise für das TEX-Programm effizienter, den Befehl `\@M` zu interpretieren, anstatt die Zeichenfolge '`10000`', die für die Zahl 10000 steht, stets neu auszuwerten.

Die Befehle werden gegenüber dem normalen Anwender gesichert, indem die Bedeutung (`\catcode`) des Zeichens '`@`' während der Definition der plain-TEX-Befehle geändert wird: Zu Beginn wird der Befehl `\catcode\@=11` gegeben, der das Zeichen '`@`' zu einem normalen Buchstaben macht. Am Ende wird durch `\catcode`\@=12` die Bedeutung wieder auf 'sonstiges Zeichen' gesetzt. Eine spätere Eingabe von '`@`' läßt dieses dann immer als 'sonstiges Zeichen' erscheinen.

Soll etwa der Befehl `\@foot`, der als letztes bei einem Befehl `\footnote` ausgeführt wird, umdefiniert werden, so wird die alte Fassung

```
\def\@foot{\strut\egroup}
```

durch

```
\catcode`\@=11
\def\@foot{\strut\bigskip\egroup}
\catcode\@`=12
```

so geändert, daß nach jeder Fußnote noch eine Leerzeile generiert wird, wobei der `\catcode`-Wert von "`@`" vor und nach der eigentlichen Definition umgesetzt wird.

Die folgenden Definitionen entsprechen der Fassung von `plain.tex` Version 3.0.

`\@cclv` Abkürzung für die Zahl 255, wird in `\pagecontents` verwendet.

```
D \chardef\@cclv=255
```

`\@cclvi` Abkürzung für die Zahl 256, verwendet in `\newlanguage`, `\newmuskip`, `\newtoks`.

```
D \mathchardef\@cclvi=256
```

`\@crfalse` ist erzeugt durch `\newif\if@cr`. Es wird verwendet in `\m@ketabbox`, das selbst durch `\+` aufgerufen wird.

```
D \def\@crtrue{\let \if@cr =\iftrue }
```

`\@crtrue` ist erzeugt durch `\newif\if@cr`. Es wird verwendet in `\m@ketabbox`, das selbst durch `\+` aufgerufen wird.

```
D \def\@crtrue{\let \if@cr =\iftrue }
```

`\@foot` ist ein Hilfsmakro in den Befehlen zu `\footnote`, aufgerufen durch `\f@@t` und `\f@t`.

```
  \def\@foot{\strut \egroup }
D \def\f@@t{\bgroup\aftergroup\@foot\let\next}
  \def\f@t#1{#1\@foot}
```

`\@if` ist ein Hilfsbefehl für die Abwicklung von `\newif`.

```
D \def\@if#1#2{\csname \expandafter \if@ \string
        #1#2\endcsname }
```

`\@ins` ist das Hilfsmakro zur Abwicklung von `\midinsert`, `\pageinsert` und `\topinsert`, wobei die eigentliche Arbeit durch `\endinsert` erledigt wird.

```
\def\@ins{\par \begingroup \setbox \z@ \vbox \bgroup }
\def\topinsert{\@midfalse\p@gefalse\@ins}
\def\midinsert{\@midtrue\@ins}
\def\pageinsert{\@midfalse\p@getrue\@ins}
```

`\@lign` ist ein Hilfsmakro zu den Befehlen `\displaylines`, `\eqalignno` und `\leqalignno`.

```
\def\@lign{\tabskip \z@skip \everycr {}}
```

`\@M` Abkürzung für die Zahl 10000, verwendet in `~`, `\break`, `\nobreak`.

```
\mathchardef\@M=10000
\def~{\penalty\@M \ }
\def\break{\penalty-\@M}
\def\nobreak{\penalty \@M}
```

`\@m` Abkürzung für die Zahl 1000, verwendet in den Befehlen `\magstep`, `\frenchspacing`.

```
\mathchardef\@m=1000
```

`\@midfalse` ist erzeugt durch `\newif\if@mid`. Es wird verwendet in den Befehlen `\topinsert`, `\pageinsert`, `\endinsert`.

```
\def\@midfalse{\let \if@mid =\iffalse }
```

`\@midtrue` erzeugt durch `\newif\if@mid`

```
\def\@midtrue{\let \if@mid =\iftrue }
```

`\@MM` Abkürzung für die Zahl 20000, verwendet in `\plainoutput`, `\supereject`, `\vfootnote`.

```
\mathchardef\@MM=20000
```

`\@ne` Abkürzung für die Zahl 1, verwendet in `\advancepageno`, `\alloc@`, `\bordermatrix`, `\mit`, `\multispan`, `\oldstyle`, `\t@bb@x`, `\tracingall`.

```
\chardef\@ne=1
```

`\@nother` ist ein Hilfsmakro zu `\settabs`.

```
\def\@nother{\dimen@ii\dimen@ \divide\dimen@ii\count@
  \setbox\tabs\hbox{\hbox to\dimen@ii{}\unhbox\tabs}%
  \advance\dimen@-\dimen@ii \advance\count@\m@ne}
```

`\@penup` wird durch `\openup` verwendet, um als Parameter eine Länge mit beliebiger Einheit zu erreichen. Die Konstruktion mit `\afterassignment` erlaubt einen Aufruf ohne Parameterklammern.

```
\def\openup{\afterassignment\@penup\dimen@=}
\def\@penup{\advance \lineskip \dimen@ \advance
       \baselineskip \dimen@ \advance \lineskiplimit
       \dimen@ }
```

`\@sf` wird als Hilfsmakro in `\footnote` neu definiert und verwendet.

\@vereq wird als Hilfsmakro in \cong verwendet.

```
\def\cong{\mathrel{\mathpalette\@vereq\sim}}
\def\@vereq#1#2{\lower .5\p@
   \vbox {\baselineskip \z@skip
          \lineskip -.5\p@
          \ialign {$\m@th #1\hfil ##\hfil $\crcr
                   #2\crcr =\crcr }}}
```

\alloc@ wird bei allen \new...-Befehlen zu Verwaltung der Zählregister verwendet.

```
\def\alloc@#1#2#3#4#5{%
    \global \advance \count 1#1by\@ne
    \ch@ck #1#4#2\allocationnumber =\count 1#1%
    \global#3#5=\allocationnumber
    \wlog {\string #5=\string
       #2\the \allocationnumber }}
\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
...
```

\c@ncel wird als Hilfsmakro für \notin gebraucht.

```
\def\notin{\mathrel{\mathpalette\c@ncel\in}}
\def\c@ncel#1#2{\ooalign
     {$\hfil #1\mkern 1mu/\hfil $\crcr $#1#2$}}
```

\ch@ck wird als Hilfsmakro bei den \new...-Befehlen verwendet, genaugenommen in \alloc@ und \newinsert.

```
\def\ch@ck#1#2#3{\ifnum \count 1#1<#2\else
   \errmessage {No room for a new #3}\fi }
```

\count@ wird als Hilfsregister bei \@nother, \hgl@, \magnification, \m@g, \newif und \s@tcols verwendet.

```
\countdef\count@=255
```

\dimen@ Abkürzung für \dimen0, verwendet durch \@nother, \@penup, \AA, \endinsert, \openup, \pagecontents. \r@@t, \s@tcols, \vgl@.

```
\dimendef\dimen@=0
```

\dimen@i Abkürzung für \dimen1, verwendet in \@nother.

```
\dimendef\dimen@i=1 % global only
```

\dimen@ii Abkürzung für \dimen2, verwendet in \@nother.

```
\dimendef\dimen@ii=2
```

\displ@y Hilfsmakro zu \displaylines, \eqalignno, \leqalignno.

```
\def\displ@y{\global \dt@ptrue
    \openup \jot \m@th
    \everycr {\noalign {\ifdt@p \global \dt@pfalse
                  \vskip -\lineskiplimit
                  \vskip \normallineskiplimit
                \else
                   \penalty \interdisplaylinepenalty
                \fi }}}
```

\dt@pfalse erzeugt durch \newif\ifdt@p, verwendet in \displ@y.

```
\def\dt@pfalse{\let \ifdt@p =\iffalse }
```

\dt@ptrue erzeugt durch \newif\ifdt@p, verwendet in \displ@y.

```
\def\dt@ptrue{\let \ifdt@p =\iftrue }
```

`\f@@t` Hilfsmakro zu Fußnotenkonstruktion, verwendet in `\fo@t`.

D
```
\def\f@@t{\bgroup \aftergroup \@foot \let \next }
```

`\f@t` Hilfsmakro zu Fußnotenkonstruktion, verwendet in `\fo@t`.

D
```
\def\f@t#1{#1\@foot }
```

`\finph@nt` ist ein Hilfsmakro zu den `phantom`-Befehlen, verwendet in den Befehlen `\makeph@nt`, `\mathph@nt`.

D
```
\def\finph@nt{\setbox \tw@ \null
   \ifv@  \ht \tw@ \ht \z@
          \dp \tw@ \dp \z@ \fi
   \ifh@ \wd \tw@ \wd \z@ \fi \box \tw@ }
```

`\finsm@sh` ist ein Hilfsmakro zu `\smash`, verwendet in den Befehlen `\makesm@sh`, `\mathsm@sh`.

D
```
\def\finsm@sh{\ht \z@ \z@ \dp \z@ \z@ \box \z@ }
```

`\fo@t` ist ein Hilfsmakro in der Fußnotenkonstruktion, aufgerufen von `\vfootnote`.

D
```
\def\fo@t{\ifcat \bgroup \noexpand \next
                 \let \next \f@@t
        \else \let \next \f@t \fi \next }
```

`\h@false` erzeugt durch `\newif\ifh@`, verwendet in `\vphantom`.

D
```
\def\h@false{\let \ifh@ =\iffalse }
```

`\h@true` erzeugt durch `\newif\ifh@`, verwendet in den Befehlen `\hphantom` und `\phantom`.

D
```
\def\h@true{\let \ifh@ =\iftrue }
```

`\hgl@` ist ein Hilfsmakro für `\hglue`, um bei der Parameterübergabe die Klammern zu vermeiden.

D
```
\def\hglue{\afterassignment\hgl@\skip@=}
\def\hgl@{\leavevmode \count@ \spacefactor
          \vrule width\z@ \nobreak
          \hskip \skip@ \spacefactor \count@ }
```

`\if@` wird bei der Konstruktion von eigenen `\if`-Anweisungen mittels `\newif\if...` verwendet:

D
```
\def\@if#1#2{%
   \csname\expandafter\if@\string#1#2\endcsname}
```

`\if@cr` wird durch `\newif\if@cr` erzeugt, verwendet in `\t@bb@x`, verändert in `\m@ketabbox`. Vorbesetzt: `\iffalse`

D
```
\newif\if@cr
```

`\if@mid` wird durch `\newif\if@mid` erzeugt, verwendet in `\endinsert`, verändert durch `\midinsert`, `\pageinsert` und `\topinsert`. Vorbesetzt: `\iffalse`

D
```
\newif\if@cr
```

`\ifdt@p` Vorbesetzt: `\iffalse`, wird nur in `\displ@y` verwendet.

D
```
\newif\ifdt@p
```

`\ifh@` Vorbesetzt: `\iffalse`, verwendet in `\finph@nt`, verändert in `\vphantom`, `\hphantom`, `\phantom`.

D
```
\newif\ifh@
```

`\ifp@ge` Vorbesetzt: `\iffalse`, verwendet in `\endinsert`, zusätzlich verändert in `\topinsert` und `\pageinsert`.

D `\newif\ifp@ge`

`\ifr@ggedbottom` Vorbesetzt: `\iffalse`, verwendet in `\pagecontents`, verändert in `\raggedbottom`, `\normalbottom`.

D `\newif\ifr@ggedbottom`

`\ifus@` verwendet in `\m@ketabbox`, verändert in `s@tt@b`, `\tabalign`. Vorbesetzt: `\iffalse`

D `\newif\ifus@`

`\ifv@` verwendet in den Befehlen `\finph@nt`, verändert in `\hphantom`, `\vphantom`, `\phantom`. Der Befehl wird mit *true* belegt, falls einer dieser drei Befehle im *vertical mode* aufgerufen wird. Vorbesetzt: `\iffalse`

D `\newif\ifv@`

`\insc@unt` Zähler für das nächste freie *insertion register*, das bei `\newinsert` belegt wird.

D `\countdef\insc@unt=20`

`\m@g` wird in `\magnification` verwendet, um die Parameterübergabe als Zuweisung zu gestalten.

```
  \def\magnification{\afterassignment\m@g\count@}
D \def\m@g{\mag \count@ \hsize 6.5truein\vsize
          8.9truein\dimen \footins 8truein}
```

`\m@ketabbox` wird in `\s@tt@b` und `\tabalign` für die Konstruktion der Tabulatorpositionen in `\settabs` verwendet.

```
  \def\m@ketabbox{\begingroup
    \global\setbox\tabsyet\copy\tabs
    \global\setbox\tabsdone\null
    \def\cr{\@crtrue\crcr\egroup\egroup
D     \ifus@\unvbox\z@\lastbox\fi\endgroup
      \setbox\tabs\hbox{\unhbox\tabsyet
                        \unhbox\tabsdone}}%
    \setbox\z@\vbox\bgroup\@crfalse
      \ialign\bgroup&\t@bbox##\t@bb@x\crcr}
```

`\m@ne` ist die Abkürzung für -1. Er wird verwendet in: `\wlog`, `\newinsert`, `\newif`, `\sp@n`, `\@nother`, `\advancepageno`, `\showhyphens`.

```
  \countdef\m@ne=22
D \m@ne=-1
  \count22
```

`\m@th` wird verwendet, wenn im Mathematiksatz eine Hilfsformel konstruiert wird, die zum umgebenden Text keinen Abstand besitzen darf. Aufgerufen in `\@vereq`, `\angle`, `\bordermatrix`, `\cases`, `\displ@y`, `\dotfill`, `\dots`, `\downbracefill`, `\eqalign`, `\leftarrowfill`, `\mathhexbox`, `\mathph@nt`, `\mathsm@sh`, `\matrix`, `\n@space`, `\r@@t`, `\rightarrowfill`, `\root`, `\underbar`, `\upbracefill`.

D `\def\m@th{\mathsurround =\z@ }`

\makeph@nt ist ein Hilfsmakro zu den phantom-Befehlen, verwendet in \ph@nt.

```
\def\ph@nt{\ifmmode\def\next{\mathpalette\mathph@nt}%
  \else\let\next\makeph@nt\fi\next}
\def\makeph@nt#1{\setbox \z@ \hbox {#1}\finph@nt }
```

\makesm@sh ist ein Hilfsmakro zu \smash.

```
\def\makesm@sh#1{\setbox \z@ \hbox {#1}\finsm@sh }
```

\mathph@nt ist ein Hilfsmakro zu den phantom-Befehlen. Es wird durch \ph@nt aufgerufen.

```
\def\mathph@nt#1#2{\setbox \z@ \hbox {$\m@th
       #1{#2}$}\finph@nt }
```

\mathsm@sh ist ein Hilfsmakro für den \smash Befehl.

```
\def\mathsm@sh#1#2{\setbox \z@ \hbox {$\m@th
       #1{#2}$}\finsm@sh }
```

\n@space wird in \big, \Big, \bigg und \Bigg verwendet und regelt den Leerraum nach einer leeren rechten Klammer (\right.) im Mathematiksatz.

```
\def\n@space{\nulldelimiterspace \z@ \m@th }
```

\p@ ist die Abkürzung für die Länge 1 pt. Die Angabe 12\p@ kostet nämlich weniger Rechenzeit und Speicher als 12pt. \p@ wird verwendet in @vereq, \Big, \Bigg, \big, \bigg, \bordermatrix, \ddots, \endinsert, \footnoterule, \makefootline, \makeheadline, \nointerlineskip, \normalbottom, \offinterlineskip, \overrightarrow, \overleftarrow, \overbrace, \raggedbottom, \underbrace, \vdots.

```
\newdimen\p@
\p@=1pt
\dimen11
```

\p@gefalse erzeugt durch \newif\ifp@ge, verwendet in den Befehlen \topinsert, \endinsert.

```
\def\p@gefalse{\let \ifp@ge =\iffalse }
```

\p@getrue erzeugt durch \newif\ifp@ge, verwendet in \pageinsert.

```
\def\p@getrue{\let \ifp@ge =\iftrue }
```

\p@renwd wird in \bordermatrix verwendet, enthält die Breite einer linken runden Klammer.

```
\newdimen\p@renwd
\setbox0=\hbox{\tenex B} \p@renwd=\wd0
```

\ph@nt ist ein Hilfsmakro zu \vphantom, \hphantom und \phantom, das die eigentliche Arbeit erledigt, nachdem \ifv@ und \ifh@ besetzt sind.

```
\def\ph@nt{%
   \ifmmode \def \next {\mathpalette \mathph@nt }%
   \else \let \next \makeph@nt \fi \next }
```

\pr@@@s ist ein Hilfsmakro zur Behandlung des Ableitungszeichens ’ im Mathematiksatz, aufgerufen durch \pr@m@s.

```
\def\pr@@@s#1{\prim@s }
```

\pr@@@t wie \pr@@@s verwendet.

```
D \def\pr@@@t#1#2{#2\egroup }
```

\pr@m@s prüft im Mathematiksatz, ob auf ein Ableitungszeichen ’ ein weiteres folgt, aufgerufen durch \prim@s.

```
  \def\pr@m@s{\ifx'\next\let\nxt\pr@@@s
D     \else\ifx^\next\let\nxt\pr@@@t
            \else\let\nxt\egroup\fi\fi \nxt}
```

\prim@s ist ein Hilfsmakro zum ’ Befehl.

```
D {\catcode`\'=\active \gdef'{^\bgroup\prim@s}}
  \def\prim@s{\prime \futurelet \next \pr@m@s }
```

\r@@t ist ein Hilfsmakro für den \root-Befehl.

```
  \def\root#1\of{\setbox\rootbox
        \hbox{$\m@th\scriptscriptstyle{#1}$}%
        \mathpalette\r@@t}
D \def\r@@t#1#2{\setbox\z@\hbox{$\m@th#1\sqrt{#2}$}
    \dimen@\ht\z@ \advance\dimen@-\dp\z@
    \mkern5mu\raise.6\dimen@\copy\rootbox
    \mkern-10mu \box\z@}
```

\r@ggedbottomfalse

erzeugt durch \newif\ifr@ggedbottom, aufgerufen durch den Befehl \normalbottom.

```
  \def\r@ggedbottomfalse
D       {\let \ifr@ggedbottom =\iffalse }
  \def\normalbottom
        {\topskip 10\p@ \r@ggedbottomfalse}
```

\r@ggedbottomtrue

erzeugt durch \newif\ifr@ggedbottom, aufgerufen durch das plain-TEX-Makro \raggedbottom.

```
  \def\r@ggedbottomtrue{\let \ifr@ggedbottom =\iftrue }
D \def\raggedbottom
      {\topskip 10\p@ plus60\p@ \r@ggedbottomtrue}
```

\rlh@ ist ein Hilfsmakro zu \rightleftharpoons.

```
  \def\rlh@#1{\vcenter
     {\hbox {\ooalign {\raise 2pt \hbox
                 {$#1\rightharpoonup $}\crcr
D         $#1\leftharpoondown $}}}}
  \def\rightleftharpoons
          {\mathrel{\mathpalette\rlh@{}}}
```

\s@tcols ist ein Hilfsmakro zu \settabs, aufgerufen durch \sett@b,

```
D \def\s@tcols#1\columns {\count@ #1\dimen@ \hsize \loop
        \ifnum \count@ >\z@ \@nother \repeat }
```

\s@tt@b ist ein Hilfsmakro zu \settabs, aufgerufen durch \sett@b.

```
D \def\s@tt@b{\let \next \relax \us@false \m@ketabbox }
```

\sett@b ist ein Hilfsmakro zu \settabs.

```
  \def\sett@b
    {\ifx \next\+\let \next \relax
D        \def\next{\afterassignment\s@tt@b\let\next}%
     \else \let\next\s@tcols
     \fi \next }
  \def\settabs{\setbox\tabs\null \futurelet\next\sett@b}
```

`\sixt@@n` ist eine abgekürzte Schreibweise für die Zahl 16, verwendet durch `\newread`, `\newwrite`, `\newfam`.

D
```
\chardef\sixt@@n=16
```

`\skip@` ist ein *skip register*, verwendet in `\vglue`, `\vgl@`, `\hglue`, `\hgl@`.

D
```
\skipdef\skip@=0
```

`\sp@n` ist ein Hilfsmakro zum `\multispan` Befehl.

```
\def\sp@n{\span \omit \advance \mscount \m@ne }
D \def\multispan#1{\omit \mscount#1
      \loop\ifnum\mscount>\@ne \sp@n\repeat}
```

`\t@bb@x` ist ein Hilfsmakro zu `\m@ketabbox`, das heißt indirekt zum Befehl `\settabs`.

```
\def\t@bb@x{\if@cr\egroup
  \else\hss\egroup \global\setbox\tabsyet
     \hbox{\unhbox\tabsyet
           \global\setbox\@ne\lastbox}%
D    \ifvoid\@ne\global\setbox\@ne\hbox to\wd\z@{}%
     \else\setbox\z@\hbox to\wd\@ne{\unhbox\z@}\fi
     \global\setbox\tabsdone
              \hbox{\box\@ne\unhbox\tabsdone}\fi
  \box\z@}
```

`\t@bbox` wie `\t@bb@x`.

D
```
\def\t@bbox{\setbox \z@ \hbox \bgroup }
```

`\thr@@` ist die abgekürzte Schreibweise für die Zahl 3, wird aber überhaupt nicht verwendet.

D
```
\chardef\thr@@=3
```

`\toks@` ist die Abkürzung für `\toks0`, wird aber gar nicht verwendet.

D
```
\toksdef\toks@=0
```

`\tw@` ist die Abkürzung für die Zahl 2, verwendet in `\bordermatrix`, `\cal`, `\finph@nt`, `\tracingall`.

D
```
\chardef\tw@=2
```

`\us@false` erzeugt durch `\newif\ifus@`, verwendet in `\s@tt@b`.

D
```
\def\us@false{\let \ifus@ =\iffalse }
```

`\us@true` erzeugt durch `\newif\ifus@`, verwendet in `\tabalign`.

D
```
\def\us@true{\let \ifus@ =\iftrue }
```

`\v@false` erzeugt durch `\newif\ifv@`, verwendet in `\hphantom`.

D
```
\def\v@false{\let \ifv@ =\iffalse }
```

`\v@true` ist erzeugt durch `\newif\ifv@`. Es wird verwendet in `\vphantom`, `\phantom`.

D
```
\def\@crfalse{\let \if@cr =\iffalse }
```

`\vgl@` ist ein Hilfsmakro zum Befehl `\vglue`, um den Längenparameter in der gleichen syntaktischen Form wie bei anderen skip-Befehlen mitzugeben.

```
\def\vgl@{\par \dimen@ \prevdepth
D    \hrule height\z@ \nobreak
     \vskip \skip@ \prevdepth \dimen@ }
\def\vglue{\afterassignment\vgl@\skip@=}
```

`\voidb@x` steht für eine permanent leere Box, verwendet in `\leavevmode`.

```
D \chardef\voidb@x=10
```

`\z@` steht für `0pt`, kann aber bei Zuweisungen auch für `0` benutzt werden; verwendet in `\advancepageno`, `\allowbreak`, `\dosupereject`, `\finsm@sh`, `\folio`, `\m@th`, `\removelastskip`, `\s@tcols`, `\underbar`.

```
D \newdimen\z@
  \z@=0pt
```

`\z@skip` ist ein *skip register* mit der Länge Null. Es wird nur einmal bei der Initialisierung `\skip\topins=\z@skip` verwendet.

```
D \newskip\z@skip
  \z@skip=0pt plus0pt minus0pt
```

14.3 Schlagwortregister

Abkürzungen durch Makros . . . 109
Absatzeinzug . . . →\parindent 21
Absatzgestaltung . . . 39ff
Absatzumbruch . . . 40
Abstand zwischen Absatzzeilen . . . 30
Abstand zwischen Absätzen . . . 27ff
Abstände in Formeln . . . 78, 151
Abstände zwischen Wörtern . . . 28
Absätze . . . 36
Abwandlung der Output-Routine . . . 175
Akzente (im Text) . . . 25
Akzente (mathematische) . . . 69
Alinea . . . 21ff
AMS-Fonts . . . 158
Anführungszeichen . . . 184
Angstrom Einheit . . . 25
Apostroph . . . 16, 184
Arbeitsmodi beim Umbruch . . . 130
Argumente bei Makros . . . 110
Aufbau der Standardseite . . . 38
Ausgabedateien . . . →\openout 194
Ausnahmelexikon (Trennung) . . . 52
Ausschließen . . . →\fontdimen 29
Balkendiagramme (Anwendungsbeispiel) . . . 179
Bedingungen in Abfragen . . . 114
Befehlsarten . . . 15
Begrenzungssymbole (mathematisch) . . . 74ff
Beispielzeile (Tabellensatz) . . . 95
Bindestriche . . . 15
Binomial-Koeffizienten (mathematisch) . . . 72
Binäre Operatoren (mathematisch) . . . 86
Bis-Strich . . . 15
Blindausgaben . . . 89
Blockklammern und Gültigkeitsbereiche . . . 14
Box-Ausgaben . . . 132
Box-Bewegungen . . . 143
Box-Dimension . . . 134
Box-Register . . . 132
Box-Teilausgaben . . . 145
Brüche (mathematisch) . . . 71
cicero (Maßeinheit) . . . 22
Computer Modern Fonts . . . 58

Datenorganisation . . . 191
Delimiter (mathematisch) . . . 74ff
Designgröße einer Schrift . . . 55
Determinante (mathematisch) . . . 80
Diakritische Zeichen . . . 25
didôt point (Maßeinheit) . . . 22
display-style . . . 70
Dollar-Zeichen . . . 14
Durchschuß (Leerraum zwischen Zeilen) . . . 30
dvi-Dateien . . . 191
Dynamischer Leerraum und Zentrierung . . . 142
Eigene Befehle . . . 109ff
Eigene mathematische Symbole . . . 154
Einfügung von Illustrationen . . . 49
Eingabedateien . . . →\input 191
Einrahmungen . . . 136
Expandierung von Makros . . . 113ff
Exponenten . . . 68
Fallunterscheidungen (mathematisch) . . . 82
Fehlermarkierungen . . . 54
Fehlermeldungen . . . 163ff
Fette Schriften . . . 56
Flatterrand . . . 33
fmt-Dateien . . . 191, 193
Fontparameter . . . →\fontdimen 29
Formatfile . . . 193
Formelausrichtung . . . 83
Formeln (hervorgehoben) . . . 70
Formeln, mehrzeilig . . . 83
Formelnumerierung . . . 84
Formelsatz . . . 65ff
Formelstände . . . 151
Formeltrennung . . . 89
Formelumbruch . . . 89
Funktionen (mathematisch) . . . 78
Fußnotenlinie . . . 50
Fußnoten . . . 50
Fußzeile . . . 38, 47
Gedankenstrich . . . 15
Gleichungen . . . 83
Globale Definitionen . . . 114
Gradzeichen (°) . . . 107
Griechische Buchstaben . . . 66ff
Große Operatorsymbole . . . 72

Grundlinien . . . 145
Gruppenklammern im Mathematiksatz . . . 65
Guillemets . . . 184
Gänsefüßchen . . . 184
Hebräische Zeichen . . . 87
Histogramme (Anwendungsbeispiel) . . . 179
horizontal mode . . . 130
Horizontale Linien . . . 53
if-Abfragen . . . 114
Illustrationen einfügen . . . 49
Indizes (mathematisch) . . . 68
Inhaltsverzeichnis . . . 194
INITEX . . . 193
integer-Register . . . →`\count` 147
Integrale (mathematisch) . . . 72
internal mode . . . 130
internal vertical mode . . . 130
Interpunktion . . . 16, 184
Kalligraphische Zeichen . . . 88
Klammern (mathematisch) . . . 74ff
Klammern, wachsende (mathematisch) . . . 77
Kleine Schriften . . . 56
Kolumnentitel . . . 47
Kommentare in der Eingabe . . . 14
Kopfzeile . . . 38, 47
Kreise . . . →`\circ` 86
Kubikwurzel . . . 69
Leeroperatoren (mathematisch) . . . 88
Leerzeichen im Text . . . 28
Ligaturen . . . 11, 185
Linien . . . 53
Linksbündige Einzelzeile . . . 17
Listengestaltung . . . 42
log-Dateien . . . 191
Längeneinheiten . . . 22
Längenregister . . . →`\dimen` 147
Makroexpandierung . . . 113ff
Makroprotokoll . . . 170
Makros . . . 109ff
Marginalien . . . 144
Markierungslinien . . . 53
Mathematiksatz . . . 65ff
Matrizen (mathematisch) . . . 79ff
Maßeinheiten . . . 22

Mehrspaltige Ausgabe . . . 176
Mehrzeilige Formeln . . . 83
Minuszeichen . . . 15
Musterzeile (Tabellensatz) . . . →\settabs 91, 95
Musterzeile . . . →\halign 95
Nationale Sonderzeichen . . . 25
Numerierung von Formeln . . . 84
Offset bei der Ausgabe . . . 37
Operatorsymbole (mathematisch) . . . 72
Output-Routinen . . . 171ff, 196
Paragraphzeichen . . . →\S 25
Parameter in Makros . . . 110
Pfeile (mathematisch) . . . 87
Phantomausgaben . . . 89
pica (Maßeinheit) . . . 22
point (Maßeinheit) . . . 22
Produktzeichen ($\prod$) . . . 73
Protokollparameter . . . 170
Prozentzeichen für den Text . . . 15
Pünktchen . . . →\dots 16
Quadratwurzel . . . 69
Rahmungen in Tabellen . . . 102
Rahmungen . . . 136
Rand einstellen . . . 36
Randausgleich . . . 33
Randmarkierungen . . . 144
Rechtsbündige Einzelzeile . . . 17
Rechtsbündige Tabulatoren . . . 93
Referenzlinie einer Box . . . 129
Relationen (mathematisch) . . . 86, 87
restricted horizontal mode . . . 130
Satzmodi . . . 129
Satzzeichen . . . 16, 184
Schriftanwahl . . . 55
Schriftenkatalog . . . 58
Schriftfamilie . . . 55
Seitenlayout . . . 21ff, 38
Seitenlänge . . . 38
Seitenunterschrift . . . →\footline 47
Seitenwechsel . . . 45ff
Seitenzählung . . . 46
Seitenüberschrift . . . →\headline 47
Senkrechte Linien . . . 53
Serifen an den Lettern . . . 55

skip-Register . . . . . →\skip 147
Slanted (schräge) Schriften 56
Sonderzeichen (AMS) 158
Sonderzeichen 25
Spaltenlinien in Tabellen 106
Spanische Satzzeichen 184
Spationieren 120
Sperren 120
Standardschriften 56
Stichwortregister 194
Summenzeichen 72
Symbole (mathematisch) 87
Tabellen mit Rahmungen 102
Tabellensatz 91ff
Tabulatoren 91
Tensorprodukt 73
Testausgaben 170
Textmarkierungen zur Referenz 177
tfm-Dateien 191
Token Register 148
Transparente Wiedergabe eines Textes 35
Trennen im Mathematiksatz 89
Trennen im Text 51
Trennmuster . . . . . →\patterns 193
Trenntabelle für INITEX 193
Trigonometrische Funktionen 78
Umbruch (seitenweise) 45
Umlaute und Trennen 52
Umlaute 24
Unterstreichen (mathematisch) 69
Variable Tabulatoren 95
Vektoren (mathematisch) 70, 87
Vereinigung (mathematisch) 73
Vergrößerungen 57
Verschiebungen mit Boxen 143
Version 3 . . . . . 16, 24, 26, 28, 40, 51,
Versmodus (zeilenweise Verarbeitung) 33
vertical mode 129
Vertikale Linien 53
Vertikaler Leerraum 27ff
Vortrenner 51
Waagerechte Linien 53
Wachsende Klammern (mathematisch) 77
Wurzeln 69

Zeilenabstand 30
Zeilenlänge . . . . . →`\hsize` 21
zentrierte Einzelzeile . . . . . →`\centerline` 17
Zentrierung im Versmodus 34
Zentrierung 142
Zweispaltige Ausgabe 146, 176
Zählregister . . . . . →`\count` 147
Überlappungen in der Textausgabe 142
Überschriften 18, 47
Überstreichen (mathematisch) 69

14.4 Fonttabellen der Standardschriften

Befehlstabelle zu den Textfonts

\Gamma	\i		0	@	P	‘	p	0
\Delta	\j	!	1	A	Q	a	q	1
\Theta	\‘	"	2	B	R	b	r	2
\Lambda	\’	\#	3	C	S	c	s	3
\Xi	\v	\$	4	D	T	d	t	4
\Pi	\u	\%	5	E	U	e	u	5
\Sigma	\=	\&	6	F	V	f	v	6
\Upsilon	[\aa]	’	7	G	W	g	w	7
\Phi	\c	(	8	H	X	h	x	8
\Psi	\ss	)	9	I	Y	i	y	9
\Omega	\ae	*	:	J	Z	j	z	A
ff	\oe	+	;	K	[	k	--	B
fi	\o	,	!‘	L	‘‘	l	---	C
ffl	\AE	-	=	M	]	m	\H	D
ffi	\OE	.	?‘	N	\^	n	\~	E
ffl	\O	/	?	O	\.	o	\"	F
0	1	2	3	4	5	6	7	

Die nebenstehende Befehlstabelle enthält die Aufrufe der Schriftzeichen, wie sie in den normalen Textfonts verwendet werden. Die Aufrufe liefern in folgenden Schriften Zeichen mit gleicher Bedeutung:

roman	10 pt	`\rm`
		`\tenrm`
	7 pt	`\sevenrm`
	5 pt	`\fiverm`
boldface	10 pt	`\bf`
		`\tenbf`
	7 pt	`\sevenbf`
	5 pt	`\fivebf`
slanted	10 pt	`\sl`
		`\tensl`
italic	10 pt	`\it`
		`\tenit`

`typewriter`
-Font

Etwas anders ist dieser Font gestaltet. Hier fehlen eine Reihe von Ligaturen und die Plätze 31 – 126 sind wie bei einer normalen ASCII-Codetabelle gestaltet.

Aufbau der Tabelle

In der Spalte 0 sind große griechische Buchstaben sowie Ligatursymbole angeordnet. Die Spalte 1 enthält die Akzentzeichen sowie nationale Sondersymbole. Der Rest der Tabelle entspricht fast vollständig einer normalen ASCII-Codetabelle, bis auf die Zeichen <, >
und \, {, }, \.

Fonttabelle zu CMR10 — Roman

Γ	ı	ˊ	0	@	P	‘	p	0
Δ	ȷ	!	1	A	Q	a	q	1
Θ	`	”	2	B	R	b	r	2
Λ	´	#	3	C	S	c	s	3
Ξ	ˇ	$	4	D	T	d	t	4
Π	˘	%	5	E	U	e	u	5
Σ	¯	&	6	F	V	f	v	6
Υ	°	’	7	G	W	g	w	7
Φ	¸	(	8	H	X	h	x	8
Ψ	ß	)	9	I	Y	i	y	9
Ω	æ	*	:	J	Z	j	z	A
ff	œ	+	;	K	[	k	–	B
fi	ø	,	¡	L	“	l	—	C
fl	Æ	-	=	M	]	m	˝	D
ffi	Œ	.	¿	N	ˆ	n	˜	E
ffl	Ø	/	?	O	˙	o	¨	F
0	1	2	3	4	5	6	7	

`\fontdimen`-Parameter

1	0.00 pt
2	3.33 pt
3	1.67 pt
4	1.11 pt
5	4.31 pt
6	10.0 pt
7	1.11 pt

Ligaturen

`ff`	ff
`fi`	fi
`fl`	fl
`ffi`	ffi
`ffl`	ffl
`!‘`	¡
`?‘`	¿
`’’`	”
`‘‘`	“
`--`	–
`---`	—

Referierung in plain-TEX: `\tenrm` (`\rm` als `\textfont0`)

Größen	*Fontdatei*	*evtl. Standardanwahl*	
5 Punkt	CMR5	`\fiverm`	(`\scriptscriptfont0`)
6 Punkt	CMR6		
7 Punkt	CMR7	`\sevenrm`	(`\scriptfont0`)
8 Punkt	CMR8		
9 Punkt	CMR9		
10 Punkt	CMR10	`\tenrm`	(`\textfont0`)
12 Punkt	CMR12		
17 Punkt	CMR17		

Fonttabelle zu CMBX10 — bold extended

Γ	ı	ˊ	0	@	P	‘	p	0
Δ	ȷ	!	1	A	Q	a	q	1
Θ	ˋ	”	2	B	R	b	r	2
Λ	´	#	3	C	S	c	s	3
Ξ	ˇ	$	4	D	T	d	t	4
Π	˘	%	5	E	U	e	u	5
Σ	¯	&	6	F	V	f	v	6
Υ	°	’	7	G	W	g	w	7
Φ	¸	(	8	H	X	h	x	8
Ψ	ß	)	9	I	Y	i	y	9
Ω	æ	*	:	J	Z	j	z	A
ff	œ	+	;	K	[	k	–	B
fi	ø	,	¡	L	“	l	—	C
fl	Æ	-	=	M	]	m	”	D
ffi	Œ	.	¿	N	ˆ	n	˜	E
ffl	Ø	/	?	O	˙	o	¨	F
0	1	2	3	4	5	6	7	

`\fontdimen`-Parameter

1	0.00 pt
2	3.83 pt
3	1.92 pt
4	1.28 pt
5	4.44 pt
6	11.50 pt
7	1.28 pt

Ligaturen

`ff`	ff
`fi`	fi
`fl`	fl
`ffi`	ffi
`ffl`	ffl
`!‘`	¡
`?‘`	¿
`’’`	”
`‘‘`	“
`--`	–
`---`	—

Referierung in plain-TEX: `\tenbf` (`\bf` als `\textfont6`)

Größen	*Fontdatei*	*evtl. Standardanwahl*	
5 Punkt	CMBX5	`\fivebf`	(`\scriptscriptfont6`)
6 Punkt	CMBX6		
7 Punkt	CMBX7	`\sevenbf`	(`\scriptfont6`)
8 Punkt	CMBX8		
9 Punkt	CMBX9		
10 Punkt	CMBX10	`\tenbf`	(`\textfont6`)
12 Punkt	CMBX12		

Fonttabelle zu CMTI10 — Italic

Γ	ı	ˋ	0	@	P	‘	p	0
Δ	ȷ	!	1	A	Q	a	q	1
Θ	ˊ	”	2	B	R	b	r	2
Λ	ˇ	#	3	C	S	c	s	3
Ξ	˘	£	4	D	T	d	t	4
Π	ˉ	%	5	E	U	e	u	5
Σ	¯	&	6	F	V	f	v	6
Υ	°	’	7	G	W	g	w	7
Φ	¸	(	8	H	X	h	x	8
Ψ	ß	)	9	I	Y	i	y	9
Ω	æ	*	:	J	Z	j	z	A
ff	œ	+	;	K	[	k	–	B
fi	ø	,	¡	L	“	l	—	C
fl	Æ	-	=	M	]	m	˝	D
ffi	Œ	.	¿	N	ˆ	n	˜	E
ffl	Ø	/	?	O	˙	o	¨	F
0	1	2	3	4	5	6	7	

\fontdimen-Parameter

1	0.25 pt
2	3.58 pt
3	1.53 pt
4	1.02 pt
5	4.31 pt
6	10.22 pt
7	1.02 pt

Ligaturen

ff	ff
fi	fi
fl	fl
ffi	ffi
ffl	ffl
!‘	¡
?‘	¿
’’	”
‘‘	“
--	–
---	—

Referierung in plain-TEX: \tenit (\it als \textfont4)

Größen	*Fontdatei*	*evtl. Standardanwahl*	
7 Punkt	CMTI7		
8 Punkt	CMTI8		
9 Punkt	CMTI9		
10 Punkt	CMTI10	\tenit	(\textfont4)
12 Punkt	CMTI12		

Fonttabelle zu CMSL10 — Slanted

Γ	ı	˙	0	@	P	‘	p	0
Δ	ȷ	!	1	A	Q	a	q	1
Θ	`	”	2	B	R	b	r	2
Λ	´	#	3	C	S	c	s	3
Ξ	ˇ	$	4	D	T	d	t	4
Π	˘	%	5	E	U	e	u	5
Σ	¯	&	6	F	V	f	v	6
Υ	°	’	7	G	W	g	w	7
Φ	¸	(	8	H	X	h	x	8
Ψ	ß	)	9	I	Y	i	y	9
Ω	æ	*	:	J	Z	j	z	A
ff	œ	+	;	K	[	k	–	B
fi	ø	,	¡	L	“	l	—	C
fl	Æ	-	=	M	]	m	”	D
ffi	Œ	.	¿	N	ˆ	n	˜	E
ffl	Ø	/	?	O	˙	o	¨	F
0	1	2	3	4	5	6	7	

\fontdimen-Parameter

1	0.16 pt
2	3.33 pt
3	1.67 pt
4	1.11 pt
5	4.31 pt
6	10.00 pt
7	1.11 pt

Ligaturen

ff	ff
fi	fi
fl	fl
ffi	ffi
ffl	ffl
!‘	¡
?‘	¿
’’	”
‘‘	“
--	–
---	—

Referierung in plain-TEX: \tensl (\sl als \textfont5)

Größen	*Fontdatei*	*evtl. Standardanwahl*	
8 Punkt	CMSL8		
9 Punkt	CMSL9		
10 Punkt	CMSL10	\tensl	(\textfont5)
12 Punkt	CMSL12		

Fonttabelle zu CMTT10 — Typewriter Type

Γ	ı	␣	0	@	P	‘	p	0
Δ	j	!	1	A	Q	a	q	1
Θ	`	"	2	B	R	b	r	2
Λ	´	#	3	C	S	c	s	3
Ξ	ˇ	$	4	D	T	d	t	4
Π	˘	%	5	E	U	e	u	5
Σ	¯	&	6	F	V	f	v	6
Υ	°	’	7	G	W	g	w	7
Φ	¸	(	8	H	X	h	x	8
Ψ	ß	)	9	I	Y	i	y	9
Ω	æ	*	:	J	Z	j	z	A
↑	œ	+	;	K	[	k	{	B
↓	ø	,	<	L	\	l	|	C
'	Æ	-	=	M	]	m	}	D
¡	Œ	.	>	N	^	n	~	E
¿	Ø	/	?	O	_	o	¨	F
0	1	2	3	4	5	6	7	

`\fontdimen`-Parameter

1	0.00 pt
2	5.25 pt
3	0.00 pt
4	0.00 pt
5	4.31 pt
6	10.49 pt
7	5.25 pt

Ligaturen

!‘	¡
?‘	¿

Referierung in plain-TEX: `\tentt` (`\tt` als `\textfont7`)

Größen	*Fontdatei*	*evtl. Standardanwahl*	
8 Punkt	CMTT8		
9 Punkt	CMTT9		
10 Punkt	CMTT10	`\tentt`	(`\textfont7`)
12 Punkt	CMTT12		

Fonttabelle zu CMMI10 — Mathematik Italic

Γ	ζ	ψ	0	∂	P	ℓ	p	0
Δ	η	ω	1	A	Q	a	q	1
Θ	θ	ε	2	B	R	b	r	2
Λ	ι	ϑ	3	C	S	c	s	3
Ξ	κ	ϖ	4	D	T	d	t	4
Π	λ	ϱ	5	E	U	e	u	5
Σ	μ	ς	6	F	V	f	v	6
Υ	ν	φ	7	G	W	g	w	7
Φ	ξ	$\leftharpoonup$	8	H	X	h	x	8
Ψ	π	$\leftharpoondown$	9	I	Y	i	y	9
Ω	ρ	$\rightharpoonup$	.	J	Z	j	z	A
α	σ	$\rightharpoondown$	,	K	$\flat$	k	$\imath$	B
β	τ	$\natural$	<	L	$\sharp$	l	$\jmath$	C
γ	υ	$\flat$	/	M	$\natural$	m	$\wp$	D
δ	ϕ	$\sharp$	>	N	$\smile$	n	$\rightarrow$	E
ϵ	χ	$\diamondsuit$	$\star$	O	$\frown$	o	$\leftarrow$	F
0	1	2	3	4	5	6	7	

\fontdimen-Parameter

1	0.25 pt
2	0.00 pt
3	0.00 pt
4	0.00 pt
5	4.31 pt
6	10.00 pt
7	0.00 pt

Befehlsnamen:

griechische Buchstaben,
siehe Seite 65
mathematische Akzente,
siehe Seite 65
Sonderzeichen,
siehe Seite 83

Referierung in plain-TEX: `\teni` (`\mit` als `\textfont1`)
automatisch im Mathematiksatz

Größen	*Fontdatei*	*evtl. Standardanwahl*	
5 Punkt	CMMI5	`\fivei`	(`\scriptscriptfont1`)
6 Punkt	CMMI6		
7 Punkt	CMMI7	`\seveni`	(`\scriptfont1`)
8 Punkt	CMMI8		
9 Punkt	CMMI9		
10 Punkt	CMMI10	`\teni`	(`\textfont0`)
12 Punkt	CMMI12		

Fonttabelle zu CMSY10 — Symbolfont

−	≍	←	′	ℵ	$\mathcal{P}$	⊢	√	0
·	≡	→	∞	$\mathcal{A}$	$\mathcal{Q}$	⊣	∐	1
×	⊆	↑	∈	$\mathcal{B}$	$\mathcal{R}$	⌊	∇	2
∗	⊇	↓	∋	$\mathcal{C}$	$\mathcal{S}$	⌋	∫	3
÷	≤	↔	△	$\mathcal{D}$	$\mathcal{T}$	⌈	⊔	4
◇	≥	↗	▽	$\mathcal{E}$	$\mathcal{U}$	⌉	⊓	5
±	≼	↘	/	$\mathcal{F}$	$\mathcal{V}$	{	⊑	6
∓	≽	≃	∣	$\mathcal{G}$	$\mathcal{W}$	}	⊒	7
⊕	∼	⇐	∀	$\mathcal{H}$	$\mathcal{X}$	⟨	§	8
⊖	≈	⇒	∃	$\mathcal{I}$	$\mathcal{Y}$	⟩	†	9
⊗	⊂	⇑	¬	$\mathcal{J}$	$\mathcal{Z}$	|	‡	A
⊘	⊃	⇓	∅	$\mathcal{K}$	∪	∥	¶	B
⊙	≪	⇔	ℜ	$\mathcal{L}$	∩	↕	♣	C
◯	≫	↖	ℑ	$\mathcal{M}$	⊎	⇕	♢	D
∘	≺	↙	⊤	$\mathcal{N}$	∧	\	♡	E
•	≻	∝	⊥	$\mathcal{O}$	∨	≀	♠	F
0	1	2	3	4	5	6	7	

`\fontdimen`-Parameter

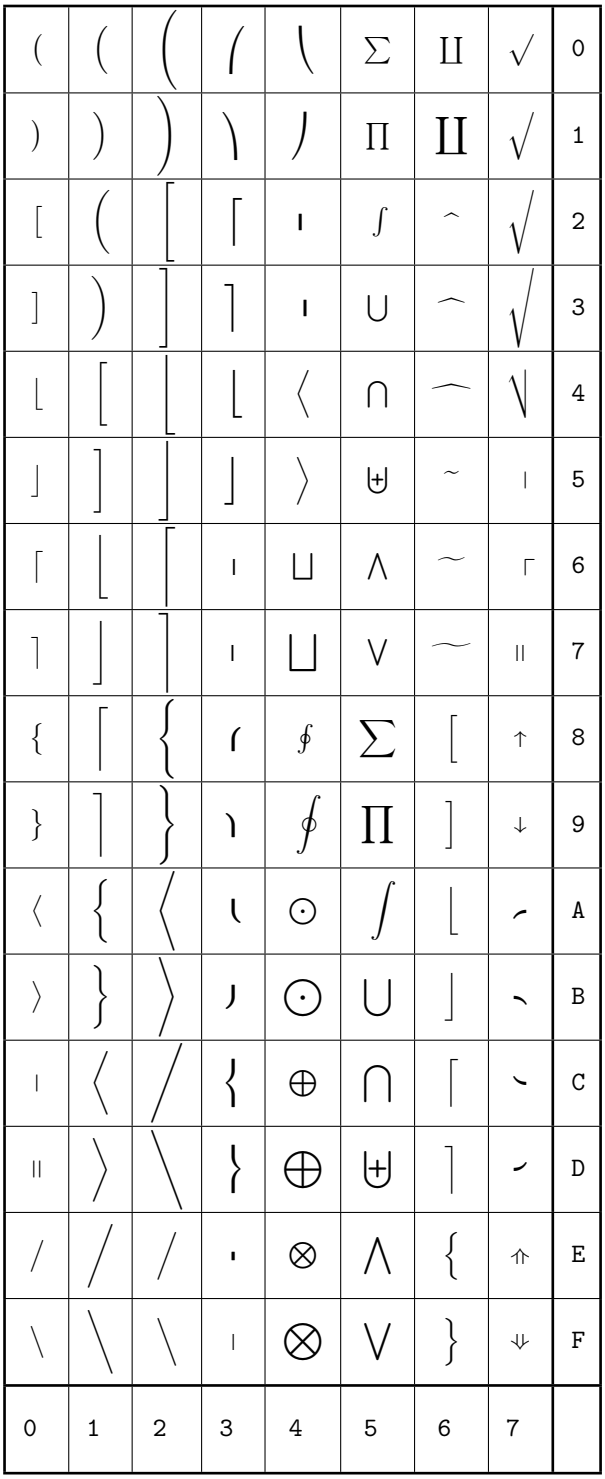
1	0.25 pt
2	0.00 pt
3	0.00 pt
4	0.00 pt
5	4.31 pt
6	10.00 pt
7	0.00 pt
8	6.77 pt
9	3.94 pt
10	4.44 pt
11	6.86 pt
12	3.45 pt
13	4.13 pt
14	3.63 pt
15	2.89 pt
16	1.50 pt
17	2.47 pt
18	3.86 pt
19	0.50 pt
20	23.90 pt
21	10.10 pt
22	2.50 pt

Befehlsnamen:

binäre Operatoren, siehe Seite 81
Relationen, siehe Seite 82
sonstige Symbole, siehe Seite 83

Referierung in plain-TEX: `\tensy` (in `\textfont2`)
automatisch im Mathematiksatz

Größen	*Fontdatei*	*evtl. Standardanwahl*	
5 Punkt	CMSY5	`\fivesy`	(`\scriptscriptfont2`)
6 Punkt	CMSY6		
7 Punkt	CMSY7	`\sevensy`	(`\scriptfont2`)
8 Punkt	CMSY8		
9 Punkt	CMSY9		
10 Punkt	CMSY10	`\tensy`	(`\textfont2`)



(	(	(	⌠	\	Σ	∐	√	0
)	)	)	\	⌡	Π	∐	√	1
[	(	[	⌈	ı	∫	⌢	√	2
]	)	]	⌉	ı	∪	⌒	√	3
⌊	[	⌊	⌊	⟨	∩	⌒	\	4
⌋	]	⌋	⌋	⟩	⊎	~	ı	5
⌈	⌊	⌈	ı	⊔	∧	~	⌜	6
⌉	⌋	⌉	ı	⊔	∨	~	‖	7
{	⌈	{	⌠	∮	Σ	[	↑	8
}	⌉	}	⌡	∮	Π	]	↓	9
⟨	{	⟨	∖	⊙	∫	⌊	⌢	A
⟩	}	⟩	⌡	⊙	∪	⌋	⌒	B
ı	⟨	/	{	⊕	∩	⌈	⌒	C
‖	⟩	\	}	⊕	⊎	⌉	⌒	D
/	/	/	ı	⊗	∧	{	↑	E
\	\	\	ı	⊗	∨	}	↓	F
0	1	2	3	4	5	6	7	

Fonttable zu CMEX10 Symbole

`\fontdimen`-Parameter

1	0.00 pt
2	0.00 pt
3	0.00 pt
4	0.00 pt
5	4.32 pt
6	10.00 pt
7	0.00 pt
8	0.40 pt
9	1.11 pt
10	1.67 pt
11	2.00 pt
12	6.00 pt
13	1.00 pt

Referierung:
`\tenex`
in `\textfont3`
automatisch

Größe: nur als CMEX10

Befehlsnamen:

Operatoren,
siehe Seite 69
Klammersymbole,
siehe Seite 71

14.5 Erweiterte 256-Zeichen TEX-Codebelegung

`	“	␣	0	@	P	‘	p	Ă	Ř	ă	ř	À	Đ	à	ð	0
´	”	!	1	A	Q	a	q	Ą	Ś	ą	ś	Á	Ñ	á	ñ	1
ˆ	„	"	2	B	R	b	r	Ć	Š	ć	š	Â	Ò	â	ò	2
˜	«	#	3	C	S	c	s	Č	Ş	č	ş	Ã	Ó	ã	ó	3
¨	»	$	4	D	T	d	t	Ď	Ť	ď	ť	Ä	Ô	ä	ô	4
˝	–	%	5	E	U	e	u	Ě	Ţ	ě	ţ	Å	Õ	å	õ	5
°	—	&	6	F	V	f	v	Ę	Ű	ę	ű	Æ	Ö	æ	ö	6
ˇ		’	7	G	W	g	w	Ğ	Ů	ğ	ů	Ç	Œ	ç	œ	7
˘	ı	(	8	H	X	h	x	Ĺ	Ÿ	Í	ÿ	È	Ø	è	ø	8
¯	ȷ	)	9	I	Y	i	y	Ľ	Ź	ľ	ź	É	Ù	é	ù	9
˙	ȷ	*	:	J	Z	j	z	Ł	Ž	ł	ž	Ê	Ú	ê	ú	A
¸	ff	+	;	K	[	k	{	Ń	Ż	ń	ż	Ë	Û	ë	û	B
˛	fi	,	<	L	\	l	|	Ň	IJ	ň	ij	Ì	Ü	ì	ü	C
‚	fl	-	=	M	]	m	}	Ð	İ	ŋ	¡	Í	Ý	í	ý	D
‹	ffi	.	>	N	^	n	~	Ő	đ	ő	¿	Î	Þ	î	þ	E
›	ffl	/	?	O	_	o	-	Ŕ	§	ŕ	£	Ï	SS	ï	ß	F
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Während der internationalen TEX-Konferenz in Cork 1990 wurde die obere Codetabelle mit einem erweiterten lateinischen Zeichensatz zusammengestellt. Die Tabelle deckt den lateinischen Schriftraum sehr gut ab. Zumindest sind praktisch alle europäischen Sonderzeichen vertreten. Wird dieser Zeichensatz verwendet, so sind die nationalen Sonderzeichen eigenständige Symbole und lösen die Trennprobleme in Sprachen mit diakritischen Zeichen.

Im europäischen Bereich sind die ersten Zeichensätze dazu in Metafont-Quellen bereits vorhanden: DC-Schriften, übrigens vom Autor dieses Buches, die nach einer gewissen Zeit der Verbesserung dann in EC-Schriften umbenannt werden.

Anmerkungen zur Belegung

Sonderzeichen

"00	`	A	Gravis	"10	“	S	» engl. openinig quotes «, deutsche Anführungszeichen oben
"01	´	A	Akut	"11	”	S	» engl. closing quotes «
"02	^	A	Zirkumflex	"12	„	S	deutsche Anführungszeichen unten
"03	~	A	Tilde	"13	«	S	franz. Anführungszeichen (guillemet), links
"04	¨	A	Trema	"14	»	S	franz. Anführungszeichen (guillemet), rechts
"05	˝	A	Doppelakut	"15	–	S	en-dash, (bis-Strich)
"06	°	A	Kringel	"16	—	S	em-dash, (Gedanken-Strich)
"07	ˇ	A	Häkchen, Háček	"17		W	» compound word mark «
"08	˘	A	Halbkreis	"18	0	T	kleine Null für Promille etc. in Kombination mit "%
"09	¯	A	Querstrich	"19	ı	T	punktloses i für Akzente
"0A	˙	A	Punkt (übergesetzt)	"1A	ȷ	T	punktloses j für Akzente
"0B	¸	A	Cedille (untergesetzt)	"1B	ff	L	f i
"0C	˛	A	Krummhaken (Ogonek)	"1C	fi	L	f f
"0D	,	S	einfaches Anführungszeichen, unten	"1D	fl	L	f l
"0E	‹	S	guillemet	"1E	ffi	L	f f i
"0F	›	S	guillemet	"1F	ffl	L	f f l

A = Akzent S = Satzzeichen L = Ligatur T = Teilzeichen

Der Bereich von "20 bis "7F entspricht der Codebelegung von ISO Latin 1. Ab "80 werden nationale und diakritische Symbole gespeichert.

"80 "A0 :	Ă ă	A, a mit Halbkreis	Rumänisch, Vietnamesisch
"81 "A1 :	Ą ą	A, a mit Krummhaken	Litauisch, Polnisch
"82 "A2 :	Ć ć	C, c mit Akut	Niedersorbisch, Obersorbisch, Polnisch, Serbokroatisch
"83 "A3 :	Č č	C, c mit Häkchen	Lettisch, Litauisch, Niedersorbisch, Obersorbisch, Serbokroatisch, Slowakisch, Slowenisch, Tschechisch
"84 "A4 :	Ď ď	D mit Häkchen, d mit Komma	Slowakisch, Tschechisch
"85 "A5 :	Ě ě	E, e mit Häkchen	Niedersorbisch, Obersorbisch, Tschechisch
"86 "A6 :	Ę ę	E, e mit Krummhaken	Litauisch, Polnisch
"87 "A7 :	Ğ ğ	G, g mit Halbkreis	Türkisch
"88 "A8 :	Ĺ Í	L, l mit Akut	Slowakisch
"89 "A9 :	Ľ ľ	L, l mit Komma	Slowakisch
"8A "AA :	Ł ł	L, l mit Schrägstrich	Niedersorbisch, Obersorbisch, Polnisch
"8B "AB :	Ń ń	N, n mit Akut	Baskisch, Niedersorbisch, Obersorbisch, Polnisch

"8C "AC	:	Ň	ň	N, n mit Häkchen	Slowakisch, Tschechisch
"8D "AD	:	Ŋ	ŋ	NJ, nj	Lappländisch
"8E "AE	:	Ő	ő	O, o mit Doppelakut	Ungarisch
"8F "AF	:	Ŕ	ŕ	R, r mit Akut	Baskisch, Niedersorbisch, Slowakish
"90 "B0	:	Ř	ř	R, r mit Häkchen	Obersorbisch, Tschechisch
"91 "B1	:	Ś	ś	S, s mit Akut	Niedersorbisch, Polnisch
"92 "B2	:	Š	š	S, s mit Häkchen	Lettisch, Litauisch, Niedersorbisch, Obersorbisch, Serbokroatisch, Slowakisch, Slowenisch, Tschechisch
"93 "B3	:	Ş	ş	S, s mit Cedille	Rumänisch, Türkisch
"94 "B4	:	Ť	ť	T mit Häkchen, t mit Komma	Slowakisch, Tschechisch
"95 "B5	:	Ţ	ţ	T, t mit Cedille	Rumänisch
"96 "B6	:	Ű	ű	U, u mit Doppelakut	Ungarisch
"97 "B7	:	Ů	ů	U, u mit Kringel	Tschechisch
"98 "B8	:	Ÿ	ÿ	Y, y mit Trema	Walisisch
"99 "B9	:	Ź	ź	Z, z mit Akut	Niedersorbisch, Obersorbisch, Polnisch
"9A "BA	:	Ž	ž	Z, z mit Häkchen	Lettisch, Litauisch, Niedersorbisch, Obersorbisch, Serbokroatisch, Slowakisch, Slowenisch, Tschechisch
"9B "BB	:	Ż	ż	Z, z mit Punkt	Maltesisch, Polnisch
"9C "BC	:	IJ	ij	IJ, ij	Niederländisch
"9D "69	:	İ	i	(!) I mit Punkt, normales i	Türkisch
"9E "D0	:	đ	Đ	(!) d, D mit Querstrich	Serbokroatisch, Vietnamesisch
"C0 "E0	:	À	à	A, a mit Gravis	Färöisch, Irisch, Isländisch, Portugiesisch, Slowakisch, Spanisch, Tschechisch, Ungarisch, Walisisch
"C1 "E1	:	Á	á	A, a mit Akut	Französisch, Gälisch, Italienisch, Katalanisch, Maltesisch, Portugiesisch, Rätoromanisch
"C2 "E2	:	Â	â	A, a mit Zirkunflex	Französisch, Portugiesisch, Rätoromanisch, Rumänisch, Türkisch, Vietnamesisch, Walisisch
"C3 "E3	:	Ã	ã	A, a mit Tilde	Portugiesisch
"C4 "E4	:	Ä	ä	A, a mit Trema	Deutsch, Estnisch, Finnisch, Schwedisch, Slowakisch, Walisisch
"C5 "E5	:	Å	å	A, a mit Kringel	Dänisch, Norwegisch, Schwedisch
"C6 "E6	:	Æ	æ	AE, ae	Dänisch, Isländisch, Norwegisch
"C7 "E7	:	Ç	ç	C, c mit Cedille	Albanisch, Baskisch, Französisch, Katalanisch, Portugisch, Türkisch
"C8 "E8	:	È	è	E, e mit Gravis	Bretonisch, Französisch, Portugiesisch, Rätoromanisch, Vietnamesisch, Walisisch
"C9 "E9	:	É	é	E, e mit Akut	Bretonisch, Französisch, Gälisch, Irisch, Isländisch, Italienisch, Katalanisch, Portugiesisch, Rätoromanisch, Slowakisch, Spanisch, Tschechisch, Ungarisch, Walisisch
"CA "EA	:	Ê	ê	E, e mit Zirkumflex	Bretonisch, Französisch, Portugiesisch, Rätoromanisch, Vietnamesisch, Walisisch
"CB "EB	:	Ë	ë	E, e mit Trema	Albanisch, Französisch, Walisisch
"CC "EC	:	Ì	ì	I,i mit Gravis	Gälisch, Italienisch

"CD "ED :	Í	í	I,i mit Akut	Färöisch, Irisch, Isländisch, Katalanisch, Portugiesisch, Slowakisch, Spanisch, Tschechisch, Ungarisch
"CE "EE :	Î	î	I,i mit Zirkumflex	Französisch, Italienisch, Rätoromanisch, Rumänisch, Türkisch, Walisisch
"CF "EF :	Ï	ï	I,i mit Trema	Bretonisch, Französisch, Katalanisch, Rätoromanisch, Walisisch
"D0 "F0 :	Ð	ð	D mit Strich, (eth)	Färöisch, Isländisch
"D0 "9E :	Đ	đ	D, d mit Strich	Serbokroatisch, Vietnamesisch
"D1 "F1 :	Ñ	ñ	N, n mit Tilde	Baskisch, Bretonisch, Spanisch
"D2 "F2 :	Ò	ò	O, o mit Gravis	Gälisch, Italienisch, Katalanisch, Rätoromanisch
"D3 "F3 :	Ó	ó	O, o mit Akut	Färöisch, Gälisch, Irisch, Isländisch, Italienisch, Katalanisch, Obersorbisch, Polnisch, Portugiesisch, Slowakisch, Spanisch, Tschechisch, Ungarisch
"D4 "F4 :	Ô	ô	O, o mit Zirkumflex	Französisch, Portugiesisch, Rätoromanisch, Slowakisch, Walisisch
"D5 "F5 :	Õ	õ	O, o mit Tilde	Estnisch, Portugiesisch
"D6 "F6 :	Ö	ö	O, o mit Trema	Deutsch, Estnisch, Finnisch, Isländisch, Rätoromanisch, Schwedisch, Türkisch, Ungarisch, Walisisch
"D7 "F7 :	Œ	œ	OE, oe	Französisch
"D8 "F8 :	Ø	ø	O, o mit Schrägstrich	Dänisch, Färörisch, Norwegisch
"D9 "F9 :	Ù	ù	U, u mit Gravis	Bretonisch, Gälisch, Italienisch, Rätoromanisch, Walisisch
"DA "FA :	Ú	ú	U, u mit Akut	Färöisch, Irisch, Isländisch, Katalanisch, Portugiesisch, Slowakisch, Spanisch, Tschechisch, Ungarisch
"DB "FB :	Û	û	U, u mit Zirkumflex	Französisch, Türkisch, Walisisch
"DC "FC :	Ü	ü	U, u mit Trema	Baskisch, Bretonisch, Deutsch, Estnisch, Französisch, Katalanisch, Portugiesisch, Rätoromanisch, Spanisch, Türkisch, Ungarisch, Walisisch
"DD "FD :	Ý	ý	Y, y mit Akut	Färörisch, Isländisch, Slowakisch, Tschechisch
"DE "FE :	Þ	þ	Thorn, thorn	Isländisch
"DF "FF :	SS	ß	Eszet	Deutsch

14.6 Literaturverzeichnis

Im Literaturverzeichnis sind ausgewählte Titel zu drei Themenbereichen aufgeführt, neben der praktischen Referenzliteratur von Knuth zu TEX überhaupt, Titel zur Buchgestaltung und Literatur, die sich mit TEX befaßt.

TEX-Referenzliteratur

Donald E. Knuth: *The TEXbook*
Volume A *'Computers and Typesetting'*, Reading, Massachusetts: Addison Wesley Publishing Company, 1990

Donald E. Knuth: *TEX: The Program*
Volume B *'Computers and Typesetting'*, Reading, Massachusetts: Addison Wesley Publishing Company, 1986

Donald E. Knuth: *The METAFONTbook*
Volume C *'Computers and Typesetting'*, Reading, Massachusetts: Addison Wesley Publishing Company, 1986

Donald E. Knuth: *METAFONT The Program*
Volume D *'Computers and Typesetting'*, Reading, Massachusetts: Addison Wesley Publishing Company, 1986

Donald E. Knuth: *Computer Modern Typefaces*
Volume E *'Computers and Typesetting'*, Reading, Massachusetts: Addison Wesley Publishing Company, 1986

Satzkunde

Jan Tschichold: *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie*
Basel: Birkhäuser, 1975

Albert Kapr: *Hundertein Sätze zur Buchgestaltung*
Leipzig: Deutsche Bücherei Leipzig, 1975

Abert Kapr: *Buchgestaltung*
Dresden: VEB Verlag der Kunst, 1963

Josef Käufer: *Das Setzerlehrbuch*
Stuttgart 1965

Barbara Salberg-Steinhardt: *Die Schrift; Geschichte — Gestaltung — Anwendung*
Köln: Dumont, 1983

TEX-Literatur

Anne Brüggemann-Klein: *Einführung in die Dokumentverarbeitung*
Stuttgart: Teubner, 1989

Wolfgang Appelt: *TEX für Fortgeschrittene*
Bonn: Addison Wesley (Deutschland), 1988

Helmut Kopka: *LaTeX — Eine Einführung*
Bonn: Addison Wesley (Deutschland), 1989

Helmut Kopka: *LaTeX — Erweiterungsmöglichkeiten*
Bonn: Addison Wesley (Deutschland), 1990

Leslie Lamport: *LaTeX: A Document Preparation System*
Reading, Massachusetts: Addison Wesley Publishing Company, 1985

Proceedings of the First European Conference on TEX for Scientific Documentation:
Reading, Massachusetts: Addison Wesley Publishing Company, 1985

Michael Spivak: *The Joy of TEX*
American Mathematical Society, Providence, RI., 1986

TEX for Scientific Documentation: *Second European Conference Strasburg, 1986*
Berlin, Heidelberg, New York: Springer, 1986

Reinhard Wonneberger: *LaTeX*
Bonn, Reading, Massachusetts: Addison-Wesley Verlag, 1987