Ein viertes Beispiel-Dokument in LaTeX

Elke Wilkeit

2003-06-29

Diese Lektion kommt mit Verspätung, tut mir Leid, da hat mich die Vorbereitung des kommenden Semesters abgelenkt. Ich hoffe, dass ihr noch nicht alles vergessen habt. Heute soll es um weitere Möglichkeiten gehen, das Layout eines Textes zu gestalten: Um Tabellen, die `quote`– und die `verbatim`–Umgebung. Ein Inhaltsverzeichnis ist am Ende dieses Textes auf Seite 5 zu finden.

1 Die `quote`–Umgebung

Der einleitende Text über dem Inhaltsverzeichnis unterscheidet sich von dem übrigen dadurch, dass er schmaler und kleiner ist. Diese Darstellung eignet sich gut für ein ***Abstract***, für einleitende Worte. Im Quelltext sieht das so aus:

```
\begin{quote}
  \small Diese Lektion kommt mit Versp"atung, ...
\end{quote}
```

Ganz einfach, nicht? Die Bezeichnung „quote“ kommt wohl daher, dass die Umgebung auch zum Zitieren längerer Textpassagen geeignet ist. Das spielt in den Geisteswissenschaften eine größere Rolle, weil dort viel zitiert wird. Aber als aufgeschlossene Menschen, die auch gelegentlich einen zusammenhängenden Text über Inhalte schreiben, zu denen auch andere schon eine Meinung geäußert haben, sollten wir uns damit natürlich auch auskennen.

2 Die `verbatim`–Umgebung

Unmittelbar einleuchtend für InformatikerInnen ist der Gebrauchswert der `verbatim`–Umgebung: Alles, was in dieser Umgebung steht, wird so ausgegeben, wie es im Quellcode steht. Das ist zum Beispiel sehr praktisch, wenn man Programmschnipsel oder auch ganze Programme in ein Dokument aufnehmen will. Ich werde sie jetzt nutzen, um zu erläutern, wie ein `Makefile` aussehen kann, das die Arbeit des Kompilierens eines LaTeX–Dokuments übernimmt.

2.1 Quellcode setzen am Beispiel des Programms `make`

Das Programm `make` wird unter Unix/Linux gern benutzt, um komplexere Programme zu kompilieren, und zwar immer nur die Teile, die inzwischen verändert worden sind. Da das `Makefile` alle Inforamtionen hat, die man zum Kompilieren des Programms braucht, kann man es mit ausliefern und damit den EndbenutzerInnen in sehr kompakter Form mitteilen, wie sie das Programm auf ihrem Rechner kompilieren können.

2.2 Ein `Makefile` für LaTeX

Nun ist LaTeX sicher nicht so komplex wie das Installationspaket für den `apache`–Server, aber nützlich kann ein `Makefile` auch da schon sein. Wer LaTeX lieber mit einem speziellen Tool verwendet, möge sich nur an dem Layout erfreuen.

```
# make Datei fuer LaTeX
# Time-stamp: <2003-06-29 17:20:36 ewi>
# Aufruf: make macht das, was hinter all: steht,
#         make psA5 macht eine ps-Ausgabe in Din A5 Format
#         make clean loescht alles, was neu erzeugt werden kann
```

```
#         make pdf macht eine pdf-Ausgabe in Din A4 Format
#         make view zeigt die dvi-Version auf dem Monitor an
# gaanz wichtig: die Befehlszeilen muessen mit einem TAB,
#               nicht mit 8 blanks, beginnen!

MAINFILE = viertes
INCLUDEFILES = vierTabellen.tex vierInhalt.tex
LATEXVIEW = xdvi

all: ${MAINFILE}.dvi

psA5: ${MAINFILE}.dvi
        dvips -o ${MAINFILE}.ps ${MAINFILE}.dvi
        psnup -2 ${MAINFILE}.ps > ${MAINFILE}A5.ps
        rm ${MAINFILE}.ps

pdf: ${MAINFILE}.tex ${INCLUDEFILES}
        pdflatex ${MAINFILE}.tex
        pdflatex ${MAINFILE}.tex
        pdflatex ${MAINFILE}.tex

view: ${MAINFILE}.dvi
        ${LATEXVIEW} ${MAINFILE}.dvi

${MAINFILE}.dvi: ${MAINFILE}.tex ${INCLUDEFILES}
        latex ${MAINFILE}.tex
        latex ${MAINFILE}.tex
        latex ${MAINFILE}.tex

clean:
        rm -f *.{aux,log}
        rm -f $(MAINFILE).{dvi,ps,pdf}
```

Ich bin keine Expertin für das Programm `make`, will hier aber trotzdem einige Worte zum Prinzip sagen. Hinter dem Schlüsselwort, z.B. `all`, steht, von welchen Dateien dieser Schlüssel abhängt. So hangelt sich make durch das Makefile:

1. Zeile 14: Für `make all` brauche ich die Datei `viertes.dvi`.
2. Zeile 29: Fur `viertes.dvi` brauche ich die Dateien `viertes. tex, vierTabellen.tex, vierInhalt.tex`.
3. Wenn die alle noch aktuell sind, tu ich gar nichts.
4. Wenn z.B. `vierTabellen.tex` geändert worden ist, muss `viertes.dvi` neu erzeugt werden und das geht, wie in den Befehlszeilen unter `$MAINFILE.dvi` beschrieben, durch dreimaliges Aufrufen von latex.

2.3 Zeilen nummerieren

Jetzt könnte ich eigentlich etwas zur Zeilen-Nummerierung von verbatim-Ausgaben sagen, aber das führt hier zu weit. Wer diese Frage vertiefen will, sei auf

`http://parsys.informatik.uni-oldenburg.de/~ewi/iub/latex/lesson7.{pdf,tex}`
`http://parsys.informatik.uni-oldenburg.de/~ewi/iub/latex/lesson1.{pdf,tex}`

verwiesen. Im ersten, einfacheren Beispiel (lesson7) sind die `verbatim`–Umgebungen alle gleich breit, im zweiten (lesson1) sind sie von variabler Breite, was mit dem Befehl `widthof{}` realisiert wurde und die Sache noch ein wenig komplizierter macht.

3 Tabellen

Ebenso wie der LaTeX– Code, der das Inhaltsverzeichnis erzeugt, steht der Quellcode dieses Abschnitts in einer gesonderten Datei, die mittels

```
\input{vierTabellen.tex}
```

in dieses Dokument eingebunden wird. Wie man sieht, erscheint der Inhalt an der betreffenden Stelle im Dokument, als wenn er hineingeschrieben worden wäre. Diese Form des Einfügens eignet sich zum Beispiel, um Grafiken oder Quellcode von Programmen einzubinden, die außerhalb des Dokuments getestet werden sollen. Der Parameter für den `input`–Befehl ist der vollständige Dateiname mit Erweiterung. Diese muss nicht `tex` heißen.

3.1 `input` oder `include`?

Wenn es darum geht, ein Buch in einzelne Kapitel einzuteilen und diese in separaten Dateien aufzubewahren, ist der `include`–Befehl noch besser geeignet. Er bekommt nur den Dateinamen, ohne Erweiterung, übergeben und beginnt den eingefügten Inhalt auf einer neuen Seite. Die Include-Datei muss die Erweiterung `tex` haben. Der Vorteil dieses Befehls ist, dass man im Vorspann mittels `\includeonly{kapitel5}` erreichen kann, dass nur die Datei `kapitel5.tex` eingefügt wird, aber die Nummerierung der Seiten und alles andere so bleibt, als wenn auch die anderen Kapitel vorhanden wären. Das macht die Arbeit an umfangreichen Dokumenten sehr viel bequemer.

3.2 Eine einfache Tabelle

Doch nun zu den Tabellen: Die einfachste Form sieht so aus:

```
\begin{tabular}{rcl}
  erste Spalte&zweite Spalte&dritte Spalte\\
  rechtsb"undig& zentriert&linksb"undig\\
  35 + 7 & = & 42\\
\end{tabular}
```

Es handelt sich um eine Umgebung, die mit `\begin{tabular}` eingeleitet wird und dahinter als die Parameter bekommt, die entscheiden, wie der Text in den Tabellenspalten angeordnet werden soll. Hier sorgt das `r` für eine rechtsbündige erste Spalte, das `c` (wie center) für eine zentrierte zweite Spalte und das `l` für eine linksbündige dritte Spalte. Die Spalten werden durch das `&` Zeichen voneinander getrennt und die Zeilen werden mit dem doppelten Backslash beendet. In der Ausgabe sieht das so aus:

erste Spalte	zweite Spalte	dritte Spalte
rechtsbündig	zentriert	linksbündig
35 + 7	=	42

3.3 Eine Spalte mit Fließtext

Die Breite einer Spalte richtet sich danach, wie viel man hineingeschrieben hat. Es gibt noch eine vierte Form des Spalten–Layouts, das ist die für fortlaufenden Text. Für diese muss explizit eine Breite vorgegeben werden. Fügen wir der Tabelle eine vierte Spalte der Breite 20 mm hinzu, indem wir die Parameterliste ergänzen auf {`rclp20mm`}, so können wir in der vierten Spalte jeweils so viel fortlaufenden Text schreiben, wie wir mögen. Beispiel:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**

3.4 Senkrechte Striche

Wenn die Tabelle in einem Kasten stehen soll, dann können wir das wie gewohnt mit einer `\fbox{}` erreichen — aber wie bekommen wir Striche in die Tabelle ***hinein***?

Senkrechte Striche werden direkt in die Parameterliste geschrieben, und zwar genau da hin, wo sie erscheinen sollen: mittels {`|rcl||p20mm|`} bekommen wir jeweils einen Strich vorne, einen hinten und zwei zwischen die dritte und vierte Spalte:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**

3.5 Waagerechte Striche

Waagerechte Striche werden mit dem Befehl `\hline` erzeugt. Wir fügen jeweils einen vor und zwei hinter der ersten Zeile ein und einen am Schluss:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**

3.6 Unterbrochene waagerechte Striche

Wenn waagerechte Striche nicht ganz durchgehen sollen, kann man mit dem Befehl `\cline{m-n}` einen Strich von Spalte m bis Spalte n ziehen, wobei $m = n$ erlaubt ist, wenn nur in einer Spalte der Strich erscheinen soll, so wie hier in der vierten:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**

3.7 Spalten zusammenfassen

Schließlich ist es noch nützlich, zu wissen, wie man über mehrere Spalten hinweg schreiben kann. Das macht man mit dem Befehl `\multicolumn`. Zum Beispiel erscheint ein Text zentriert unter die ersten drei Spalten, wenn in der Tabelle an der betreffenden Stelle steht

```
\multicolumn{3}{c}{(drei Spalten von dynamischer Breite)}
```

Dieser Befehl ersetzt die ersten drei Spalten in der Zeile und dahinter geht es dann mit Trennzeichen & und der vierten Spalte weiter. So sieht es aus:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**
(drei Spalten von dynamischer Breite)			

Nanu, da fehlt doch etwas? Im Layout der Dreierspalte hätten wir die senkrechten Striche mit eintragen müssen. Aber das ist schnell nachgeholt:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**
(drei Spalten von dynamischer Breite)			

3.8 Unterbrochene senkrechte Striche

Damit ist nun auch schon geklärt, wie wir es erreichen können, dass ein senkrechter Strich nicht ganz durchgeht: Wir können dafür den Befehl `\multicolumn` verwenden, denn der muss sich nicht unbedingt auf mehr als eine Spalte beziehen. Mit

```
\multicolumn{1}{l}{35 + 7}
```

erreichen wir z.B., dass der Text 35 + 7 linksbündig gesetzt wird, obwohl er in einer ansonsten rechtsbündigen Spalte steht. Außerdem ist der senkrechte Strich am Anfang der Tabelle dann unterbrochen:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**
(drei Spalten von dynamischer Breite)			

3.9 Aufgaben

1. Der unterbrochene Strich am Anfang der Tabelle sollte durchgehend sein. Wie ist das zu erreichen? Kopiere den Quelltext der Tabelle und korrigiere ihn.

2. Die drei ersten Spalten sollen über einige aber nicht alle Zeilen durch senkrechte Striche getrennt sein, so, wie es unten zu sehen ist. Bitte schau nicht in den Quelltext, sondern versuche selbst, die folgende Ausgabe zu bekommen:

erste Spalte: r	zweite Spalte: c	dritte Spalte: l	vierte Spalte: p{40mm}
rechtsbündig	zentriert	linksbündig	Hier kann nun massenhaft fortlaufender Text stehen.
35 + 7	=	42	**Dieser fette fortlaufende Text steht in der dritten Zeile.**
(drei Spalten von dynamischer Breite)			

Inhaltsverzeichnis