# 2. LaTeX

In dieser Aufgabe erstellen Sie ein kleines LaTeX-Dokument und erzeugen daraus eine PDF-Datei.

**a)** Öffnen Sie ein Terminalfenster und erzeugen Sie darin mit

```
mkdir tex; cd tex
```

ein neues Verzeichnis `tex`, in das Sie direkt hinein wechseln. Starten Sie dann mit Eingabe von `gedit &` den Texteditor; Sie starten mit einer leeren, unbenannten Datei, und durch das angehängte „&"-Zeichen bleibt die Shell benutzbar.

Machen Sie sich zunächst mit den Tastenkombinationen vertraut, die Sie in der Ulix-Devel-VM benötigen, um die Zeichen `\` (Backslash), `{` und `}` (geschweifte Klammern) sowie `[` und `]` (eckige Klammern) einzugeben – diese werden Sie beim Bearbeiten von TeX-Dateien häufig benötigen.

Wer die VM unter Windows oder Linux installiert hat, sollte die fünf Sonderzeichen über die üblichen Tastenkombinationen erreichen können.

Anwender von Mac OS können die folgenden Shortcuts verwenden:
[Alt+8] = `[`, [Alt+9] = `]`, [Alt+7] = `{`, [Alt+0] = `}`, [Alt+ß] = `\`. (Alle Kombinationen jeweils mit der rechten (!) Alt-Taste.) Auch interessant: [Alt+<] = `|`, [Alt+Q] = `@`.

**b)** Geben Sie nun im Terminalfenster die folgenden vier Zeilen ein:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\begin{document}
\end{document}
```

Speichern Sie die Datei, indem Sie [Strg+S] drücken und als Dateinamen z. B. `uebung02.tex` angeben. Gleich nach dem Speichern sollte sich das Syntax Highlighting aktivieren, da der Editor jetzt weiß, dass Sie eine TeX-Datei bearbeiten.

Fügen Sie zwischen `\begin{document}` und `\end{document}` ein wenig Text ein; verwenden Sie dabei auch deutsche Umlaute. In der Shell können Sie (nach dem erneuten Speichern mit [Strg+S]) nun mit dem Kommando

```
pdflatex uebung02.tex
```

aus der `tex`- eine `pdf`-Datei machen und diese anschließend mit

```
xpdf uebung02.pdf
```

im PDF-Viewer betrachten. (Sie verlassen den Viewer mit [Strg+W].)

Probieren Sie anschließend die in der Vorlesung vorgestellten Gestaltungsmöglichkeiten aus: Testen Sie insbesondere die Abschnittsnumerierung (mit `\section`, `\subsection` und `\subsubsection`), die Textauszeichnungen fett und kursiv (`\textbf{...}`, `\emph{...}`) sowie numerierte Listen und Bullet-Listen (über `enumerate`- und `itemize`-Umgebungen).

Wenn Sie den PDF-Viewer auch mit angehängtem &-Zeichen starten, bleibt das Terminal weiter benutzbar. Nach dem Neu-„Kompilieren" der `tex`-Datei mit `pdflatex` erkennt der PDF-Viewer selbständig, dass sich die PDF-Datei geändert hat, und aktualisiert die Ansicht.

Schließen Sie am Ende die Editor- und PDF-Viewer-Fenster.

## 3. Literate Programming: Code lesen und „tangeln“

Rufen Sie das Skript `update-ulix.sh` auf – im Ordner `tex/` finden Sie danach einen Unterordner `wc/` mit zwei Dateien (`wc.pdf` und `wc.nw`). Sollte das Skript nicht funktionieren, finden Sie die beiden Dateien auch unter http://faq.ktug.org/wiki/pds/noweb/wc.pdf und http://www.rpi.edu/locker/79/000679/noweb/2.7/examples/wc.nw .

**a)** Wechseln Sie in den Ordner `tex/wc/` in Ihrem Home-Verzeichnis und lesen Sie die PDF-Datei: Sie beschreibt als Literate Program die Implementierung des Unix-Tools `wc` (word count), das zu einer oder mehreren gegebenen Datei(en) die Anzahl der Zeichen, Wörter und Zeilen ausgibt. Verwendet das Literate Program einen *Top-down*- oder einen *Bottom-up*-Ansatz?

**b)** Jetzt „tangeln“ (sprich: „tängeln“) Sie den kompilierbaren Quelltext: Geben Sie den Befehl

```
notangle wc.nw > wc.c
```

ein und öffnen Sie anschließend die aus dem Literate Program extrahierte C-Quellcode-Datei `wc.c` im Editor – vergleichen Sie den Code mit dem Literate Program. Im obigen Aufruf von `notangle` mussten Sie keinen Code Chunk angeben, denn in der Datei ist ein Chunk namens ⟨*⟩ definiert, dessen Definition auf Seite 2 der PDF-Datei steht: Dieser ist der „Standard-Chunk“, wenn Sie keinen über `-R`*`chunkname`* angeben.

Können Sie anhand der C-Datei noch erkennen, ob das Programm *top-down* oder *bottom-up* entwickelt wurde?

**c)** Probieren Sie auch die Option zum Erzeugen von Zeilennummern aus, das geht mit

```
notangle -L wc.nw > wc.c
```

Betrachten Sie wieder wc.c im Editor. Sie finden nun zahlreiche Zeilen der Form `#line 111 "wc.nw"`: Diese führen dazu, dass bei Kompilierfehlern mit dem C-Compiler (`gcc`) nicht die Nummer der fehlerhaften Zeile in der C-Datei, sondern die in der `nw`-Datei ausgegeben wird.

**d)** Übersetzen Sie das Programm durch Eingabe von `gcc wc.c` – es erscheint eine Warnung (*wc.nw:146: warning: incompatible implicit declaration of built-in function ‘exit’*). Prüfen Sie, dass in der Datei `wc.nw` der Befehl `exit(status);` in Zeile 146 steht. Sie können das (trotz der Warnung) erzeugte Programm testen, indem Sie

```
./a.out wc.nw
```

(mit führendem Punkt und Schrägstrich) eingeben; Sie sollten die folgende Ausgabe erhalten:

```
     377    1895   12405 wc.nw
```

## 4. Code in ein Literate Program konvertieren

Nun ist es Zeit, das Literate Programming selbst auszuprobieren. Dazu können Sie eine klassisch programmierte und dokumentierte C-Quelldatei `shell.c` verwenden, die automatisch im Ordner `tex/` gelandet ist, als Sie in Aufgabe 3 das Skript `update-ulix.sh` aufgerufen haben.

**a)** Lesen und verstehen Sie das Programm, das eine kleine Shell implementiert.

**b)** Konvertieren Sie das Program in ein Literate Program `shell-lp.nw`. Zerlegen Sie es dazu in geeignete Chunks, die Sie gut dokumentieren. Klassische C-Kommentare sollen keine überbleiben. Eine Template-Datei `shell-lp.nw` ist bereits vorbereitet.

**c)** „Tangeln“ Sie (wie in Aufgabe 3 b und c) aus `shell-lp.nw` die C-Datei `shell-lp.c` und vergleichen Sie diese mit dem Original.

**d)** Erzeugen Sie mit `noweave -index -delay shell-lp.nw > shell-lp.tex` die TeX-Datei, rufen Sie dann zweimal `pdflatex shell-lp.tex` auf und betrachten Sie das Ergebnis mit `xpdf shell-lp.pdf`.