Computational Physics - Eine Einführung

Kurzreferenz

Helge Todt & Thorsten Tepper García

Institut für Physik und Astronomie

Version vom 9. März 2012

Inhaltsverzeichnis

Bedeutung der Schrifttypen:

`xvzf`	wörtlich einzugebender Text
argument	Platzhalter für selbst zu ersetzenden Text
[. . .]	optionaler Teil eines Befehls
befehl	UNIX/Linux-Befehl
STRG	Taste, die gedrückt werden soll

Teil I.
UNIX/Linux

1. Der Computerpool

Der Lehrcomputerpool befindet sich im Raum 0.087 im Haus 28 (Institut für Physik und Astronomie), der Studentencomputerpool im Raum 1.100 im selben Haus. Derzeit gibt es folgende Rechner (Hostnames):

Raum 0.087		Raum 1.100
NFS-Server	nachts aus	SSH-Login möglich
born	**bohr**	**bizet**
boyle	**berry**	**brahms**
grieg	**biot**	**chopin**
hertz	**bloch**	**liszt**
joule	**hahn**	**plato**
volta	**haydn**	**ravel**
weber	**mozart**	**vivaldi**
	niels	**wagner**
	nobel	
	tesla	

`.stud.physik.uni-potsdam.de`

Im Raum 0.087 gibt es den PostScript-Drucker **strauss** und im Raum 1.100 den PostScript-Drucker **mach**, welcher ein Seitenlimit von 500 Seiten pro Nutzer und Semester eingestellt hat.

Für Nutzer ohne eigenen Account gibt es auf jedem Rechner einen lokalen (!) User-Account, dessen Logindaten auf dem linken Whiteboard im Raum 0.087 stehen.

2. Shell

2.1. Dateibefehle

Bei den Befehle **chmod**, **chown**, **cp**, **ln**, **mv**, **rm** und **scp** haben folgende Optionen dieselbe Bedeutung:

-f force: keine Nachfrage (übertrumpft **-i**)
-i interaktiv: Nachfragen
-r, -R rekursiv: beziehe Verzeichnisse mit ein

cat *datei*

gibt *datei* auf STDOUT aus.
Beispiel: Konkatenieren von Dateien
```
cat datei2 >> datei1
```

cd [*verzeichnis*]

Verzeichniswechsel. Ohne Argument Wechsel ins Homeverzeichnis.
cd - Wechsel ins vorige Verzeichnis.

chmod [**-Rf**] *modus datei*

Änderung der Zugriffsrechte auf *datei*. Es gibt zwei verschiedene Modi. Am einfachsten zu merken ist der Modus, der Buchstaben verwendet, Setzen der Rechte mit **-** Entziehen der Rechte mittels **+**

- **u** User (Eigentümer der Datei)
- **g** Gruppe
- **o** others: alle anderen
- **a** alle: sowohl **u** als auch **g** als auch **o**
- **r** readable: lesbar
- **w** writeable: beschreibar
- **x** executable: ausführbar (Programme, Skripte, Verzeichnisse)

Beispiel:
```
chmod -R go-rwx ~/.ssh
```
entzieht Gruppe und anderen alle Rechte für Verzeichnis ~/.ssh und dessen Inhalt (rekursiv **-R**).
Die Rechte können alternativ auch direkt mittels einer dreistelligen Oktalzahl gesetzt werden. Dabei steht die erste Stelle für den User, die zweite für die Gruppe und dritte für alle anderen. Die einzelnen Stellen sind jeweils die Summe der vergebenen Zugriffsrechte:

- **0** keinerlei Zugriffsrechte
- **1** executable: Ausführungsrechte
- **2** writeable: Schreibrechte
- **4** readable: Leserechte

Beispiel:
```
chmod 750 Dokumente
```
Setzt alle Rechte für den User, Lese- und Ausführungsrecht für die Gruppe und keinerlei Rechte für alle anderen.

chown [**-Rf**] [*user*][*:gruppe*] *datei*

Setze User- und Gruppeneigentümer für *datei*.
Beispiel:
```
chown -R tepper:atp Downloads
```

cp [**-frpL**] *quelle ziel*

Kopieren von Dateien/Verzeichnissen.
Optionen:

-p preserve: Behalte Dateirechte und Zeiten bei.
-L Link dereferenzieren (statt Link kopieren)

Beispiel:
```
cp -r ~/mail/ private/.
```

df [**-h**]

Zeigt den verfügbaren und belegten Speicherplatz auf den Festplatten an. Ausgabe in GByte usw. mittels **-h**.

du [**-hs**]

Ermittle Speicherplatzbedarf des aktuellen Verzeichnisses und seiner Unterverzeichnisse/Dateien. Ausgabe in GByte usw. mittels **-h**, mit **-s** (short) nur Speicherplatz des gesamten Verzeichnisses.

file *datei*

Zeige Dateityp von *datei* an.

find *verzeichnis kriterium optionen*

Sucht in *verzeichnis* und dessen Unterverzeichnissen Dateien, die *kriterium* entsprechen und führt dann Aktion *optionen* aus. **Kriterien:**

-name *name* Dateien mit Namen *name*, reguläre Ausdrücke müssen mit **" "** geschützt werden

-iname *name* wie **-name**, aber ignoriert Groß-/Kleinschreibung

-mtime *n* Dateien, die in den letzten $n \times 24$ h geändert wurden

grep [**-ABCilmnrv**] *muster* [*datei[en]*]

sucht nach *muster* in *datei[en]* oder in STDOUT, falls hinter einer Pipe.

-A *n* zeigt *n* Zeilen nach Fundort an

-B *n* zeigt *n* Zeilen vor Fundort an

-C *n* wie **-A** und **-B** zusammen

-i ignore case: unterscheide nicht zw. Groß- u. Kleinschreibung

-l zeige nur Namen der Dateien, in denen *muster* gefunden wurde

-m *n* stoppe Lesen nach *n*tem Fund in Datei

-n zeige auch Zeilennummer des Fundorts

-r rekursiv: auch Dateien in Unterverzeichnissen

-v invert: zeige nur Zeilen, die nicht *muster* enthalten

less *datei*

Zeigt den Inhalt einer Datei seitenweise an (sog. Pager), daher auch recht nützlich wenn rechts von einer Pipe, z.B.

```
ls -l | less
```

Blättern in `less` kann man mit der Leertaste (seitenweise) aber auch mit den Cursortasten (zeilenweise).

ln [**-sf**] *datei linkname*

Anlegen eines Links. Symbolische Links (**-s**) können über Verzeichnisse hinweg angelegt werden.

ls [**-aclrt**] [*pfad*]

Listet den Inhalt eines Verzeichnisses auf. Häufigste Optionen:

-a all: alle, auch versteckte Dateien (mit **.** am Anfang) anzeigen

-c zusammen mit **-l**: Zeit der letzten Veränderung

-h human readable: Größe in besser lesbaren Einheiten (MByte, GByte, usw.)

-l long format: Zugriffsrechte, Anzahl der Links, Besitzer, Gruppe, Größe, Zeit

-r reverse: kehre Sortierreihenfolge um

-t time: nach Zeit sortieren

-S size: nach Größe sortieren

-u zusammen mit **-l**: Zeit des letzten Zugriffs

Beispiel:
ls -altr zeigt alle Dateien nach Zeit umgekehrt sortiert an

mkdir [**-p**] *verzeichnis*

Anlegen eines neuen Verzeichnisses. Die Option **-p** erzeugt bei Angabe eines Pfades auch die entsprechenden Oberverzeichnisse (parental directories), bzw. gibt keine Fehlermeldung, falls das Zielverzeichnis schon existiert.

more *datei*

Zeigt den Inhalt einer Datei seitenweise an (sog. Pager), ähnlich wie → `less`, aber nur seitenweises Blättern.

mv *quelle ziel*

Verschieben oder Umbenennen von Dateien oder Verzeichnissen.

pwd

Gibt den absoluten Pfad zum aktuellen Verzeichnis zurück. Vgl. auch Shell-Variable `PWD`.

rm [**-ifr**] *datei*

Löschen (remove) von Dateien bzw. Verzeichnissen (mit Option **-r** für rekursiv). Es können auch mehrere Dateien/Verzeichnisse angegeben werden.

rmdir

Löschen von leeren Verzeichnissen.

tar *aktion archiv* [*datei*]

Tape archiver zum Zusammenpacken von Dateien und Verzeichnissen. Aktionen:

- **c** erzeuge *archiv* (create) aus Datei/Verzeichnis *datei*
- **f** Archiv ist ein file (default: Bandlaufwerk)
- **t** zeige Inhalt von Archiv
- **v** zeige ausgeführte Aktionen (verbose)
- **x** extrahiere Archiv
- **z** Archiv ist gegzippt (**.tar.gz** bzw. **.tgz**)

Beispiel:
tar xvzf paket.tar.gz
Entpackt das Archiv `paket.tar.gz`.

touch *datei*

Anlegen einer leeren bzw. ändern des Zeitstempels einer bestehenden Datei.

2.2. Login und Datenaustausch

hostname [**-d**]

Ausgabe des Rechnernamens bzw. der Domäne (Option **-d**).

rsync [**-actvz**] [*quelle*] [*ziel*]

Remote-Copy über verschlüsselte Verbindung (ssh, scp), mit der Möglichkeit nur geänderte Dateien zu übertragen.

- **-a** Archiv-Modus=**-rlptgoD**
- **-c** checksum: geänderte Dateien an Checksumme erkennen (statt Datum und Größe)
- **-D** Beibehaltung von speziellen Dateien und Geräten
- **-g** group: Beibehaltung der Gruppe
- **-l** links: Kopiere symbolische Links (keine Derefenrenzierung)
- **-o** owner: Beibehaltung des Besitzers (nur als root)
- **-p** permissions: Beibehaltung von Zugriffsrechten
- **-r** rekursiv: auch Unterverzeichnisse
- **-t** time: behalte Modifikationszeiten bei
- **-v** verbose: Ausgabe weiterer Informationen
- **-z** komprimiere Dateien für Übertragung

Beispiel:

rsync -tvzc * joule:lehre/.
Übertrage nur die geänderten (regulären) Dateien des aktuellen Verzeichnisses zum Host **joule** nach ~**/lehre/**

scp [**-Cpr**] [*quelle*] [user@][*host*]:[*ziel*]

scp [**-Cpr**] [user@][*host*]:[*quelle*] [*ziel*]

Remote-Copy über verschlüsselte Verbindung.

-C compress: komprimierte Datenübertragung
-p preserve: behalte Dateiattribute (Zeit) bei
-r recursive: kopiere Verzeichnisse

ssh [*user@*] host [**-Xvvv**] [*befehl*]

Verschlüsselte Verbindung (secure shell - SSH) zu entferntem Rechner *host*. Falls möglich, wird durch **-X** eine Weiterleitung von grafischen Fenstern (X11) zum lokalen Rechner erlaubt. Debug-Ausgabe durch **-v**, **-vv** oder **-vvv** (volle Debug-Ausgabe). Statt eines interaktiven Logins kann auch *befehl* ausgeführt werden.

ssh-keygen

generiert ein SSH-Schlüsselpaar

su [**-**] [*user*]

Substituiere User. Anmeldung als anderer *user*, mit **-** wird dessen Login-Skript ausgeführt, oder (ohne Argument) als **root**.

sudo *befehl*

Führe *befehl* als **root** aus, **root**-Passwort erforderlich.

w [**-f**]

Gibt Liste der zur Zeit auf dem Rechner eingeloggten User aus. Mit **-f** (from) auch angezeigt, von wo aus sich diese eingeloggt haben.

whoami

Gibt den Namen (ID) des aktuellen Users aus.

2.3. Shell und Prozesse

befehl1 **|** *befehl2*

Pipesymbol, verbindet STDOUT des Befehls links vom Pipesymbol (*befehl1*) mit STDIN des Befehls rechts vom Pipesymbol (*befehl2*).
Beispiel:
`ls -l | grep "^d"`
Zeigt alle Verzeichnisse im aktuellen Verzeichnis an.

befehl **>** *datei*

Umleitungsoperator, lenkt STDOUT von *befehl* in Datei *datei* um. Existiert die Datei bereits, so wird sie überschrieben. Soll statt dessen angehängt werden, muss >> verwendet werden, z.B.
`date >> anmeldung.log`
Soll STDERR umgeleitet werden (bash): `befehl 2>` *datei*
Soll STDOUT und STDERR umgeleitet werden (bash): `befehl 2>` *datei* `2>&1`

&

Ausführen eines Jobs im Hintergrund, wird an den entsprechenden Befehl angehängt.

STRG + c

Sendet an den im Terminal laufenden Prozess das Signal TERM, um diesen zu beenden.

STRG + d

Shell beenden (**exit**).

STRG + q

Beendet das Pausieren der Shell, falls STRG + s vorher gedrückt wurde.

STRG + s

Shell suspenden (pausieren).

alias [name[=wert]] (bash)

alias [name[wert]] (tcsh)

Ausgabe aller Aliase, des Aliases mit Namen *name* oder Setzen des Aliases mit Namen *name* auf *wert*.

bg [*job*]

Führt *job* oder (ohne Argument) den zuletzt gestarteten Job im Hintergrund der Shell fort, insbesondere nach <STRG>+z (Prozess pausieren), Jobnummer wird in eckigen Klammern in der ersten Spalte angezeigt.

echo [**-n**] [*text*]

Gibt *text* auf STDOUT aus. Mit **-n** unterbleibt der abschließende Zeilenumbruch.

exit

Beendet die laufende Shell-Sitzung.

export [*VARIABLE=wert*] (bash)

Setzt die Umgebungsvariable (Vererbung auf alle Tochtershells) *Variable* und initialisiert sie mit dem Wert *wert*. Ohne Argumente werden alle gesetzten Umgebungsvariablen angezeigt.

fg [*job*]

Bringt *job* wieder in den Vordergrund, bzw. (ohne Argument) den zuletzt gestarteten Job. Siehe **bg** .

history [**-c**]

Zeigt die zuletzt eingebenen Befehle an. Löschen der History mit **-c**.

kill [**-signal**] *PID*

Sendet *signal* an Prozess mit der Prozess-ID *PID* (kann mit **top** oder **ps** angezeigt werden). Die beiden wichtigsten Signale sind **TERM** bzw. **15**, welches den Prozess regulär beendet und **KILL** bzw. **9**, welches den Prozess sofort beendet (kein Abspeichern o.ä.).

nice [**-n *inc***] *programm*

Starte *programm* mit herabgesetzter Priorität. Falls kein Inkrement *inc* gegeben, wird Priorität um $+10$ geändert. Superuser darf auch negatives Inkrement angeben. Prozessprioritäten im Bereich -20 (höchste Priorität) bis $+20$ (niedrigste Priorität). Rechenjobs sollten stets mit niedrigster Priorität gestartet werden (vor allem auf älteren UNIX-Systemen), um den Rechner nicht zu blockieren.

ps [**axu**]

Anzeige laufender Prozesse. Ohne Argumente werden nur die Prozesse des aktuellen Terminals angezeigt. Die Optionen **axu** lässt **ps** alle Prozesse auf dem Host zusammen mit den Namen der User anzeigen.

set [*variable=wert*] (tcsh)

variable=wert (bash)

Setzen der lokalen Variablen *variable* und Zuweisung von *wert*. Ohne Argumente werden alle gesetzten lokalen Variablen angezeigt.

setenv [*VARIABLE wert*] (tcsh)

Setzt die Umgebungsvariable (Vererbung auf alle Tochtershells) *Variable* und initialisiert sie mit dem Wert *wert* Ohne Argumente werden alle gesetzten Umgebungsvariablen angezeigt.

unset *variable*

unsetenv *VARIABLE* (tcsh)

Löschen der Variablen *variable*. Man beachte, dass auf der **`tcsh`** der Befehl **`unset`** nur lokale, aber keine Umgebungsvariablen löscht.

top

Interaktive Anzeige laufender Prozesse, folgende Eingaben sind nach dem Start möglich:

1		Zeige Load für jeden Prozessorkern einzeln an
`i`		idle: Zeige nur laufende, keine, schlafenden Prozesse
`k`	***PID***	kill - Beende Prozess *PID*
`q`		quit - Beenden
`u`	***user***	Zeige nur Prozesse von *user*

Eine verbesserte Version von `top` ist das Programm `htop` .

which [*befehl*]

Zeigt den Pfad zum Programm *befehl* an, bzw. ob es sich um einen shellinternen Befehl oder um einen Alias (einige Shells, z.B. tcsh) handelt.

xterm [**`-ls`**]

Öffnet ein neues Xterm. Mit **`-ls`** wird die Shell darin zu einer Login-Shell.

2.4. Informationen

apropos *keyword*

Listet Befehle auf, in deren Beschreibung *keyword* auftaucht

id [*user*]

Ausgabe der User-ID und Gruppenzugeörigkeit.

info *befehl*

Gibt Informationen zu *befehl* aus, wie **`man`**

man *befehl*

Gibt Informationen zu *befehl* aus, wie **`info`**

/proc/, z.B. **cat /proc/cpuinfo**

Virtuelles Verzeichnis, in welchem Informationen über Prozesse und Hardware abgelegt werden.

uname [**`-a`**]

Informationen über das Betriebssystem (Kernelversion). Mit **`-a`** werden alle Informationen ausgegeben. Auf SuSE-Linux-Systemen findet man die Distributionsversion in der Datei `/etc/SuSE-release`.

2.5. Drucken

a2ps [-o *output*] *datei*

Wandelt ASCII-Datei *datei* nach PostScript um, sendet die Ausgabe an den Standarddrucker, es sei denn mittels **-o** ist eine Ausgabedatei spezifiziert.

lpr [-P*drucker*] *datei*

Druckt *datei* auf *drucker*

lpq [**-p*drucker***]

Gibt Informationen zu Druckjobs für *drucker* aus.

lpstat [**-tl**] [**-p*drucker***]

Gibt Informationen zum Status von *drucker* aus, mit **-tl** erfolgt eine detaillierte Ausgabe.

3. Linux-Anwendungsprogramme

cal [*monat*] [*jahr*]

Ausgabe einer Kalenderansicht für den aktuellen Monat, einen angegebenen Monat oder ein ganzes Jahr.

acroread [*datei.pdf*]

Startet den Acrobat Reader zum Anzeigen und Ausdrucken von pdf-Dateien.

firefox [*datei.htm[l]*]

Der Standard-Webbrowser.

gv [*datei.ps*]

Anzeigeprogramm für PostScript-Dateien (**.ps**).

konqueror [*datei.htm[l]*]

Datei- und Webbrowser des KDE. Nicht auf allen Rechner verfügbar.

nautilus

Dateibrowser des GNOME-Desktops.

4. Editoren

4.1. vi

vi [*datei*]

Der **vi** unterscheidet zwischen Einfüge- und Befehlsmodus. Er läuft in jedem Terminal und darf daher nicht im Hintergrund (mit **&**) gestartet werden. Meist läuft statt des `vi` der `vim`, bei dem man auch im Einfügemodus die Cursortasten benutzen kann.

i Wechseln in den Einfügemodus

a Wechseln in den Einfügemodus (Anhängen)

ESC Wechseln in den Befehlsmodus

:w [*datei*] Schreiben (in Datei) - im Befehlsmodus

:q Verlassen - im Befehlsmodus

:q! Verlassen (ohne Nachfrage) - im Befehlsmodus

x Ein Zeichen Löschen - im Befehlsmodus

dd Eine ganze Zeile löschen

4.2. nedit

nedit [*datei*]

Windowsartiger Editor, der spaltenweise mit der Maus markieren kann und auf vielen Systemen verfügbar ist.

STRG + *Linksklick* Spaltenweises Markieren mit der Maus.

STRG + c Markierten Bereich kopieren

STRG + v Kopierten Bereich einfügen

STRG + s Datei speichern

STRG + q Beenden

4.3. emacs

emacs [**-nw**] [*datei*]

Der **emacs** ist auf fast jedem Unix-System verfügbar und kann unter MacOS nachinstalliert werden (DarwinPorts), wobei es auch Implementierungen für die jeweiligen Grafischen Oberflächen gibt (**xemacs** für X11, **aquamacs** für MacOS).
Die Option **-nw** (no window) lässt den **emacs** im Terminal laufen (ähnlich dem **vi**). Die u.g. Tasten sind ihrer Beschriftung (deutsches Layout) entsprechend durch STRG usw. bezeichnet. Sollen zwei Tasten gleichzeitig gedrückt werden, so ist dies durch ein + angezeigt, ansonsten sind die Tasten nacheinander zu betätigen.

STRG + g letzten Befehl abbrechen

STRG + x STRG + s Speichern des aktuellen Buffers (Fensterinhalt)

STRG + x STRG + c Schließen (close) des Editorfensters

STRG + k Ausschneiden ab Cursor bis Zeilenende (kill)

STRG + y Einfügen des kopierten/ausgeschnittenen Bereichs (yank)

STRG + x r k Spaltenweises Ausschneiden des markierten Bereichs (rectangular kill)

STRG + x r y Spaltenweises Einfügen des markierten Bereichs (rectangular yank)

STRG + SPACE Bereich Markieren Anfang. Cursor mit Pfeiltasten (Cursortasten) bis zum Ende des Bereichs bewegen.

STRG + w Ausschneiden des markierten Bereichs

ESC w Kopieren des markierten Bereichs

STRG + s Suchen eines Begriffs im Dokument

ESC g Gehe zur Zeile (Zeilennummer)

STRG + x i Eine Datei einfügen

Teil II.
C/C++

5. Grundlagen in C/C++

5.1. Dateien und Kompilieren

In C++ ist die Trennung von Deklaration in Datei *name***.h** und Definition in Datei *name***.cpp** üblich.

g++ [**-c**] [**-g**] [**-I***dir*] [**-L***dir*] [**-l***lib*] [**-o** *ausgabe*] [**-O***level*] [**-W***warnung*] *quelldatei*

Kompilieren mit dem GNU C++ Compiler, Optionen:

-c compile: nur Kompilieren, nicht Linken, Ergebnis ist *name*`.o`
-g debug: Einfügen von Debugsymbolen für GDB (Gnu-Debugger)
-I *verzeichnis*: suche auch in *verzeichnis* nach Header-Dateien
-L *verzeichnis*: suche auch in *verzeichnis* nach Bibliotheken
-l *library*: linke gegen Bibliothek *library*, z.B. `-lX11`
-o output: Name der zu erzeugenden Datei, typischerweise ein Programm
-O *[level]* Optimieren, verschiedene Stufen: 0, 1, 2, 3
-W *warnung*: Warne, verschiedene Fälle, z.B. `-Wall` (alles)

Die Art des Kompilierens (ggf. auch Linken) richtet sich nach Endung der Quelldatei (z.B. Linken bei `.o`-Dateien). **Beispiel:**

```
g++ -o name beispiel.cpp
```

5.2. Programmstruktur

Alle Prozeduren von C/C++ sind Funktionen (d.h. mit Rückgabewert, ggf. **void**), so auch das eigentliche Programm (Funktion **main**).

```
#include <iostream>
using namespace ::std ;

int main()
{
    Anweisung ;
    return 0 ;
}
```

{ **Anweisung1** ; **Anweisung2** ; }

Block durch geschweifte Klammernt **{}** zusammengehalten. Wo eine Anweisung steht, kann auch ein Block stehen.

/* **Kommentar** */

Kennzeichnung eines Kommentarbreiches, beliebig viele Zeilen.

// **Kommentar**

Rest der Zeile ist ein Kommentar (nur C++)

5.3. Bibliotheken:

In C/C++ selbst sind nur Strukturen implementiert, z.B. `for`-Schleifen, der ganze Rest, z.B. Eingabe/ Ausgabe, wird über Bibliotheken ermöglicht.

#include <bibliothek>

Einbinden der entsprechenden Bibliothek. Namen von Standardbibliotheken werden mit spitzen Klammern angegeben. Beispiele:
#include <**fstream**>
#include <**cmath**>

5.4. Deklarationen

typ variable **;**

Variablendeklaration für Variable vom Typ *typ* und mit Namen *variable*, sollte (muss aber nicht) am Anfang eines Blocks erfolgen und ist dann lokal (sichtbar) bezüglich dieses Blocks. Mögliche Typen siehe Sect. 5.12, 5.35, 5.36. Beispiel:

```
double x ;
```

typ name **(** *typ argument1, . . . , typ argumentN* **) ;**

Funktionsdeklaration einer Funktion mit Namen *name* und Rückgabwert vom Typ *typ*. Funktionen müssen vor *main()* deklariert werden, um darin verwendet werden zu können. Beispiel:

```
double square (double x) ;
```

5.5. Lokale Variablendefinitionen

Variablendefinitionen sind i.d.R. lokal und gelten nur für den aktuellen Block `{ }`. Sie sind standardmäßig `auto`, d.h. mit Blockeintritt werden die Variablen neu belegt. Soll ihr Inhalt für den nächsten Blockeintritt erhalten bleiben, so müssen sie als `static` deklariert werden.

```
void meineFunktion () { static int counter ; ... }
```

Lokale Variablendefinitionen dürfen auch in der **for**-Schleife erfolgen und gelten nur in diesem Block z.B.

```
for ( int i = 1 ; i < 15 ; i++ ) ... ;
```

5.6. Globale Variablen

Werden vor *main()* deklariert und sind in allen nachfolgenden Funktionen und nach außen (andere Quelldateien) sichtbar. Bei Verdeckung durch lokale Variablen gleichen Namens, kann mittels vorangestellter Doppelpunkte `::` die globale Variable angesprochen werden. Beispiel:

```
int i = 3 ;
main()
{
  int k, i = 2 ;
  k = i + ::i ; // Ergebnis ist 5
}
```

Globale Variablen, die `static` deklariert wurden, sind nicht nach außen sichtbar.

5.7. Inline Funktionen

inline *typ name* (*arg*)

Kurze Funktionen können mit dem Schlüsselwort *inline* vor *main()* vereinbart werden. Beispiel:

```
inline double quadrat(double x) { return x*x ; }
```

Der Aufruf im Programm erfolgt dann z.B. mit `quadrat(7.2) ;`

5.8. Funktionsstruktur - CALL BY VALUE

C/C++ unterstützt mehrere Arten der Variablenübergabe. Wird eine normale Variable als Argument angegeben, so wird lediglich deren Wert übergeben (kopiert), die Variable selbst kann von der gerufenen Funktion nicht verändert werden.

```
double eigene_funktion(int k) // k ... formaler Parameter
{
   double z = 2.3 ;           // lokale Variable
   return (z * k) ;           // Rückgabewert festlegen
}
```

5.9. Initialisierung von Funktionsparametern

Die Parameter einer Funktion können in C++ sofort initialisiert werden:

```
void funktion ( int a = 10, int b = 100, float c = 0.1 ) {...}
```

Falls die Funktion mit fehlenden Parametern aufgerufen wird, werden die Initialisierungen übernommen.

5.10. Einführung von Referenztypen

Referenztypen werden mit dem Referenzoperator & (hier nicht zu verwechseln mit dem Adressoperator) eingeführt, sie müssen grundsätzlich bei der Deklaration initialisiert (d.h. einem Ziel zugeordnet) werden.

```
int a = 20 ;
int &b = a ;
b = 50 ;
```

Die Referenz ist einfach als ein Alias-Name zu verstehen. Die Initialisierung kann nicht wieder geändert werden. Auf sie wird wie eine normale Variable (ohne Dereferenzierungsoperator) zugegriffen.

5.11. Parameterübergabe CALL-BY-REFERENCE

Die Parameter werden als Referenzen übergeben. Beispiel:

```
void swap(int &x, int &y) {...} ;
```

Aufruf:

```
int w2, w1 ;
...
swap ( w2, w1) ;
```

Wenn die Funktion verlassen wird, bleiben die in der Funktion manipulierten Werte dann erhalten (Referenzen).

5.12. Datentypen

Die angegebenen Größen beziehen sich auf die üblichen Einstellungen auf 64Bit-Systemen. Diese können z.B. durch Compiler-Optionen verändert werden.

Ganzzahltypen:

Typ	Größe	Zahlenbereich
`short`	2 Byte	-32768 bis +32767
`unsigned short`	2 Byte	0 bis +65535
`int`	4 Bytes	-2147483648 bis +2147484647
`unsigned int`	4 Bytes	0 bis +4294967295
`long`	8 Bytes	-9.2×10^{18} bis $+9.2 \times 10^{18}$
`unsigned long`	8 Bytes	0 bis $+18\,446\,744\,073\,709\,551\,615$
`char`	1 Byte	ASCII-Zeichen (-128 bis 127)

Gleitkommatypen:

Typ	Größe	Mantisse	Exponent
`float`	4 Bytes	7-8 Stellen	-45 . . . 38
`double`	8 Bytes	15-16 Stellen	-324 . . . 308
`long double`	10 Bytes	19-20 Stellen	-4951 . . . 4932

5.13. Deklaration und Initialisierung

Variablen dürfen bei ihrer Deklaration sofort initialisiert werden.

```
int a = 7 ;
```

5.14. Schlüsselwort const

const *typ = wert* **;**

Das Schlüsselwort *const* in einer Deklaration definiert eine Variable als konstant. Sie muss sofort initialisiert werden und ihr Wert kann dannach nicht mehr geändert werden. Beispiel:

```
const int = 5 ;
```

Vorteil: Erlaubt bestimmte Compileroptimierungen.
Variablen vor Veränderungen schützen, auf die mit einem Pointer zugegriffen wird:

```
const int *pj = &j ;
pj* = 3 ; // Compilezeitfehler
```

Deklarationen von Funktionsparametern mit `const` sorgen dafür, dass die Funktion die Parameter

nicht ändern kann:

```
kubus (const double &x) ... ; // Referenzübergabe, nur lesend
```

5.15. Vordefinierte Konstante

Bei Verwendug der Mathematikbibliothek

```
#include <cmath>
```

sind einige mathemtischen Konstanten vordefiniert, z.B.:

`M_PI`	π	`M_PI_2`	$\frac{\pi}{2}$
`M_E`	e	`M_SQRT2`	$\sqrt{2}$

5.16. Statische Feldvereinbarung

typ name [*dim*] *[dim2]...*

Felder (Arrays) von beliebigen Datentypen können bei der Deklaration mittels eckiger Klammern `[]` statisch alloziert werden. Beispiele:

```
double y[4][3] ; // zweidimensionales Feld, Matrix (double)
int z[17] ;      // eindimensionales Feld, Vektor (int)
char name[10] ;  // eindimensionales Feld für Characters (String)
```

5.17. Dynamische Feldvereinbarung

Es ist auch erlaubt, lokale Arrays variabler Länge zu deklarieren:

```
void my_function (int dim1, int dim2)
{
     int matrix[dim1][dim2] ;  // zweidimensionales Array
                               // variabler Länge
      Anweisungen ;
}
```

5.18. Arithmetische Operatoren (1)

+, **-**, *****, **/**

Werden von den jeweiligen Bibliotheken auch überladen.

5.19. Arithmetische Operatoren (2)

% (Modulo)

Beispiel: `rest = k % m ;`

++ (Inkrementieren)

Beispiel: `zaehler ++ ;`

-- (Dekrementieren)

Beispiel: `countdown-- ;`

5.20. Logische Operatoren

`<`, `>`, `==`, `<=`, `>=`, `!=`

Logische Operatoren für Vergleiche. Ergebnis eines Vergleichs ist ein logischer Ausdruck (Wert `true` oder `false`)

&& (und), **||** (oder)

Verküpfung von logischen Ausdrücken

5.21. Kontrollstrukturen

break ;

Verlassen einer Schleife oder eines `switch-case`-Blocks und Sprung zum Ende des Blocks, z.B.

```
if ( a < b ) break ;
```

continue ;

Abbruch des aktuellen Schleifendurchlaufs und Sprung zum Beginn der Schleife.

do *anweisung* ; **while** *logausdr*

Die `do`-Schleife wird solange ausgeführt, wie die Bedingung am Ende der Schleife (nach `while`) wahr ist, allerdings immer mindestens einmal, da die Prüfung erst am Ende der Schleife erfolgt, z.B.

```
do x = x + 2. ; while ( x < 0. ) ;
```

for (*var* = *init* ; *logausdr* ; *ausdr*) *anweisung* ;

Zählsschleife, änlich einer `while`-Schleife mit zusätzlichen Schleifensteuerungsmechanismen. Vor dem ersten Semikolon im Schleifenkopf steht die Initialisierungsanweisung, danach ein zu prüfender Ausdruck und hinter dem zweiten Semikolon ein Ausdruck der nach jedem Schleifendurchlauf ausgewertet wird. Beispiele:

```
for ( k = 0 ; k < 6 ; k++ ) sum = sum + 7 ;
for ( double z = 0.7 ; z < 17.2 ; z = z +0.3 ) {...}
```

if *logausdr anweisung*

Bedingte Ausführung von Anweisungen (oder Blöcken `{...}`) falls *logausdr* wahrt ist. Es kann auch in einen alternativen Block mittels `else` verzweigt werden. Beispiele:

```
if (a == 0) cout << "Ergebnis " << endl ;
if (a == 0) a = x2 ; else a = x3 ;
if ( x < M_PI )
{ return ; }
else if ( x > 4)
{ y = 3. ; }
```

switch *ausdr* **case** *wert* : *anweisung* ;

Mehrfache Auswahl. Der Ausdruck *ausdr* muss eine Ganzzahl (auch `bool` oder `char`), ergeben. Nach jedem erfolgreiche angesprungenen `case` wird auch der nächste ausgeführt, sodass ein `break` am Ende eines jeden `case`-Blockes notwendig ist, falls dies nicht gewünscht ist.

```
switch (ausdruck)
{
      case  wert1 :  Anweisung ; break ;
      case  wert2 :  Anweisung ; break ;
      case  wert3 :  Anweisung ; break ;
      default :  anweisung ;
}
```

while *logausdr anweisung*

Eine `while`-Schleife wird solange ausgeführt, wie die Bedingung im Schleifenkopf wahr ist. Ist die Bedingung schon beim Schleifeneintritt `false`, so wird der Schleifenkörper nie ausgeführt (übersprungen).

```
while ( x < 0. ) x = x + 2. ;
```

5.22. Explizite Typumwandlung (casts)

In der Regel wird der Compiler bei Operationen zwischen verschiedenen Typen den kleineren Typ (z.B. `int`) in den größeren umwandeln (z.B. `double`), sog. implizite Konvertierung.

var2 = **(typ2)** var1 ;

var2 = **typ2 (**var1**)** ;

Der Variablen *var2* wird der Wert von *var1* zugewiesen und dabei umgewandelt, sodass der Wert dem Datentyp von *var2* entspricht. In C++ darf der Name des Typs auch als Funktion zum Konvertieren genutzt werden.

```
int i = 2 ;
double x, y;
x = double (i) ; // optional, sonst implizit durch Compiler
y = 1 / (double) i ; // notwendig, ansonsten Ergebnis y = 0
```

Darüberhinaus ist es auch möglich, Strings in Zahlendatentypen umzuwandeln (u.U. ist dafür das Inkludieren von `stdlib.h` erforderlich):

atoi (const char *str) ;

Umwandlung eines C-Strings in eine Ganzzahl (Typ `int`), z.B.
`iarg = atoi ( argv[1] ) ;`

atof (const char *str) ;

Umwandlung eines C-Strings in eine Gleitkommazahl (Typ `double`), z.B.
`x = atof ( line ) ;`

5.23. Ein-/Ausgabe in C++ (streams)

Mittels `#include <stdio.h>` können alle in C bekannten Ein- und Ausgabefunktionen (z.B. `printf`, `scanf`) in C++ weiterhin verwendet werden.
Die grundlegenden Ein- und Ausgabemethoden in C++ werden durch Inkludieren von `<iostream>` verfügbar. Durch Definition des zu verwendenen Namensraumes:

```
using namespace ::std ;
```

können die entsprechenden Funktionen ohne den Scope-resolution-Operator `::` genutzt werden:

cin *>> arg1 [>> arg2 ... >> argn]* ;

Einlesen von der Kommandozeile in Argument(e) *arg1 ... argn* (Variablen).

cout *<< arg1 [<< arg2 ... << argn]* ;

Ausgabe von Argument(en) *arg1 ... argn* (Variablen, Literale) auf STDOUT (Kommandozeile).

Beispiel

```
cout << "Bitte eine Zahl eingeben: " ;
cin  >> x ;
cout << "Bitte einen String eingeben: " ;
cin  >> str ;
cout << "Sie haben " << x << " und "
     << str << " eingegeben." << endl ;
```

Im Unterschied zu den C-Funktionen wird der vereinbarte Datentyp bei `cin` usw. nicht explizit angegeben (Überladung).

Es funktionieren die schon aus C bekannten *Escape-Sequenzen:*

`\t`	horizontaler Tabulator	`\r`	Wagenrücklauf (carriage return)
`\b`	Backspace	`\"`	Anführungszeichen
`\n`	neue Zeile (entspricht `endl`)	`\\`	Backslash

Anstelle der Argumente können bei `cout` auch ***Manipulatoren*** eingesetzt werden, z.B.:
`fixed, scientific, left, right, internal, dec, oct, hex, showbase, showpoint, uppercase, showpos.`

Durch Einbinden der Bibliothek <`iomanip`> stehen weitere Manipulatoren zur Verfügung:
`setprecision(n)` ... Runden auf `n` Stellen. Beispiel:

```
cout << setprecision(4) << 3.14159 << endl ;
```

Ergibt `3.142`.

5.24. Dateien anlegen und lesen

Binden Sie die Bibliothek <*fstream*> und <*iostream*> ein.
Folgende wichtige Klassen stehen nun zur Verfügung:

- die Klasse **ofstream** für die Ausgabe in eine Datei
- die Klasse **ifstream** für die Eingabe aus einer Datei
- die Klasse **fstream** für Ein- und Ausgabe

5.25. Schreiben von Daten in eine Datei

Bibliothek inkludieren: <`fstream`>

```
ofstream dateiout ; // Objekt der Klasse ofstream anlegen
dateiout.open("sinuskurve.dat") ; Methode open der Klasse ofstream
dateiout << x ;  // Auslesen der Daten
dateiout.close() // Schließen der Datei
```

5.26. Lesen von Daten aus einer Datei

Bibliotheken inkludieren: <`fstream`>:

```
char buffer[100]
ifstream dateiin ; // Objekt der Klasse ifsteam anlegen
dateiin.open("sinuskurve.dat") ;
while ( dateiin.good() )   // good überprüft u.a., ob das
                              Dateiende erreicht ist
{
      dateiin.getline(buffer,100) ; // Zahlen einlesen,
                                       Buffer (100) festlegen
      cout << zahlen << endl ;    // Zeichen auf Bildschirm ausgeben
      dateiin.close() ;
}
```

5.27. Pointer (Zeiger)

typ name* ;

Deklaration eines Pointers `name` für den Datentyp `typ`.

Pointervariablen gibt es für jeden Datentyp (auch für Pointer), sie enthalten die Speicheradresse einer Variable des jeweiligen Datentyps.
Pointerdeklaration:

```
float *z, a = 2.1 ; // Gleitkommapointer und Gleitkommavariable
```

Pointerinitialisierung:

```
z = & a ; // z enthält jetzt die Adresse von a
```

Ausgabe der in einer Zeigervariablen gespeicherten Adresse:

```
cout << "Adresse des Pointers z: " << z ;
```

Wert der Variablen `a` ändern:

```
*z = 5.2 ; // * ist der Dereferenzierungsoperator
```

5.28. Parameterübergabe an Funktionen mit Zeigern

Neben CALL-BY-VALUE und CALL-BY-REFRENCE gibt es noch die Möglichkeit der Übergabe von Zeigern an Funktionen, z.B.:

```
void swap(int *a, int *b) //Pointer als formale Parameter
{
     int tmp ;
     tmp = *a ;
      *a = *b ;
      *b = tmp ;
}
```

Aufruf dieser Funktion in `main()`:

```
swap(&x, &y) ;   // Adressen werden übergeben
```

5.29. Strukturen und Klassen

struct *name* { ... } ;

class *name* { ... } ;

Definition einer Struktur oder Klasse `name`, d.h. einer Sammlung von Variablen und Funktionen (Member und Methoden). Beispiel:

```
struct pkw
{  float geschwindigkeit ;
   int   leistung ;
   int   zylinder ;
   void oeffnetuer (bool &tuer) { tuer = true ; }
   private:
   char  kraftstoff ;
};
```

Strukturvariable definieren und initialisieren:

```
struct pkw mercedes = {195.0, 125, 8, "Super" } ;
```

Wertzuweisung an Strukturkomponenten:

```
mercedes.geschwindigkeit = 200.0 ;
```

Auf mit `private` versehene Member kann von außen nicht zugegriffen werden. Per Default sind alle Member in Klassen `private` und in Strukturen `public`.

Aufruf einer Methode:

```
mercedes.oeffnetuer(tuer1) ;
```

Zeiger auf Strukturen:

```
struct pkw *ptpkw ;
```

Zeiger initialisieren:

```
ptpkw = &mercedes ;
```

Zugriff auf den Wert eines Elements der Struktur:

```
(*ptpkw).geschwindigkeit = 180.5 ;
```

Äquivalente Schreibweise dafür ist:

```
ptpkw -> geschwindigkeit = 180.5 ;
```

Arrays von Strukturvariablen:

```
struct pkw auto_flotte[100] ;
```

5.30. Definition eigener Datentypnamen mit typedef

typedef *typ name* ;

Definition eines neuen Datentypnamens *name* vom Typ *typ* (Aliasen). Es existiere beispielsweise die Struktur:

```
struct maschinendaten ;
```

Dann kann diese mit dem Befehl `typdef` in das kürzere `mdat` umbenannt werden:

```
typedef struct maschinendaten mdat ;
```

Weiteres Beispiel: s. Sect. 5.35.

5.31. Aufzählungstypen mit enum

enum *name* {...}

Definition des Aufzählungsdatentyps enum:

```
enum name_des_aufzählungstypen
{  name_1,
   name_2,
   ...
   name_n
};
```

Die Einträge sind intern durchnummeriert, z.B.

```
enum Tag {Mo, Di, Mi, Do, Fr, Sa, So} ;
Tag heute, morgen, gestern ; // Deklaration von Tagen
heute = Mi ; // Zuweisung
if (heute == So) cout << "freier Tag" ; // Vergleich
if (heute > Fr) cout << "Wochendende!" ; // Reihenfolge
```

5.32. Ausgewählte Funktionen aus der Bibliothek <cmath>

Prototyp: double funktion (double)

```
cos()  ; sin()  ; tan()  ;
asin() ; atan() ; acos() ;
cosh() ; sinh() ; tanh() ;
exp()  ; fabs() ;
log()     ; // natürlicher Logarithmus (Basis e)
log10()   ; // dekadischer Logarithmus (Basis 10)
pow(x,y)  ; // x^y
sqrt(a)   ; // Quadratwurzel aus a
fmod(a,b) ; // Rest der Ganzzahldivision a / b
```

5.33. Zufallszahlengenerator

rand()

Zufallszahlengenerator. Erzeugt ganzzahlige Pseudo-Zufallszahlen im Bereich 0 bis RAND_MAX. Erfodert Inkludieren der <cstdlib>.
Initialisierung erfolgt mittels der Funktion srand(unsigned), z.B. Initialisierung mit der Zeit (#include<ctime>):

```
srand((unsigned)time((long *) 0)) ;
```

Gleitkommazahlen zwischen 0 und 1 könnte man mittels

```
r = rand() / (RAND_MAX + 1.0) ;
```

erzeugen.
Der Zufallsgenerator drand48() erzeugt Pseudo-Zufallszahlen vom Typ double zwischen 0 und 1. Initialisierung erfolgt mittels srand48(unsigned). Initialisierung mit der Zeit (#include<ctime>):

```
srand48((unsigned)time((long *) 0)) ;
```

5.34. Übergabe von Kommandozeilenargumenten (Parameter) an Programme

```
int main ( int argc, char *argv[])
```

Im Funktionskopf von *main* werden zwei formale Parameter angegeben:

`argc` Anzahl der übergebenen Argumente (*argument-count*)
Programmname zählt als erstes Argument

`argv` Array der übergebenen Argumente (*argument-vector*)

Beispiel: Starten des Programms `progam` mit Argument `param1`:

```
./program param1
```

Alternative Schreibweise: `int main ( int argc, char **argv )`

5.35. Komplexe Zahlen

Durch Einbinden der Standardbibliothek

```
#include <complex>
```

können Datentypen für komplexe Zahlen definiert werden, z.B.

```
typedef std::complex<double> complex_d ;
```

sodass entsprechende komplexe Variablen definiert werden können:

```
complex_d z ;
z = complex_d ( 1.2, 3.4) ;
```

Die entsprechenden Operatoren (z.B. $+$) und Funktionen (z.B. `cos()`) können dank Überladung weiterhin genutzt werden.

5.36. Boolscher Datentyp

In C++ steht gibt es einen boolschen Datentyp mit den Literalwerten `true` und `false`, die identisch `1` bzw. `0` sind.

```
bool bset = true ;
```

Teil III.
LaTeX / TeX

LaTeX erlaubt das Erstellen von beliebigen Dokumenten, z.B. Journalartikel, Masterarbeiten, Bücher, Präsentationen, Briefe oder Poster.

6. Dokumente erstellen

TeX ist ein von Donald E. Knuth entwickeltes Textsatzsystem, das automatisch den Text gemäß typografischen Regeln formatiert. LaTeX ist eine *paketbasierte* Makroerweiterung für TeX von Leslie Lamport. LaTeX verwendet sog. Dokumentklassen, die bestimmte Voreinstellungen für Dokumente enthalten.

6.1. PDFs mit LaTeX erzeugen

pdflatex *datei*.tex

Direktes Erzeugen eines PDF-Dokuments aus einer LaTeX-Datei. Es werden weitere Dateien für Referenzen, Inhaltsverzeichnis usw. erzeugt.

latex *datei*.tex

Erzeugen einer DVI-Datei aus einer LaTeX-Datei. Diese kann mittels `dvips` in eine PS-Datei umgewandelt werden. Notwendig bei der Verwendung von PSTricks.

dvips *datei*.dvi

Umwandeln einer DVI-Datei in eine PostScript-Datei.

ps2pdf *datei*.ps

Umwandeln einer PS- in eine PDF-Datei.

6.2. Dokumentstruktur

TeX-Dateien bestehen aus einer Präambel, in der dokumentweite Einstellungen vorgenommen werden und dem eigentenlich Text nach `\begin{document}` bis `\end{document}`. **Beispiel:**

```
\documentclass[DIV12,11pt]{scrartcl} % DIN A4
% Praeambel, Kommentar
\usepackage[english,german]{babel} % Tabelle statt table usw.
\usepackage[utf8]{inputenc} % direkte Eingabe von Umlauten

\begin{document}

 Hallo, Welt!

\end{document}
```

Befehle werden mit einem Backslash \ eingeleitet und können Pflichtargumente in geschweiften Klammern { } und optionale Argumente in eckigen Klammern [ ] haben.
In der ersten Zeile wird mittels `\documentclass{ }` die Dokumentklasse festgelegt, z.B. `scrartcl` (deutsche Version von `article`), siehe auch Abschnitt 12.
Das Prozentzeichen % leitet einen Kommentar ein.

6.3. Probleme beim Kompilieren

Erkennt `pdflatex` oder `latex` einen Befehl nicht, so wird gestoppt und es erscheint z.B. folgende Fehlermeldung:

```
\! Undefined control sequence.
l.6 \nd
       {document}
?
```

Mittels `x` + Eingabetaste kann die Übersetzung abgebrochen werden. Das Nicht-Erkennen von Befehlen kann zwei Ursachen haben:

- fehlerhafte Schreibweise, im obigen Bsp. etwa `\nd{document}` statt des korrekten `\end{document}`
- das benötigte Paket, welches den Befehl bereitstellt, wurde nicht eingebunden, s. Abschnitt 6.4

6.4. Pakete einbinden

Pakete, z.B. `babel`, werden mittels

```
\usepackage[german]{babel}
```

eingebunden. In eckigen Klammern stehen Paketoptionen. Pakete erweitern die Funktionalität von LaTeX.
Beispiele:

`babel` mit Option `german` für deutsche Bezeichnungen, also “Inhaltsverzeichnis” statt “table of contents”

`inputenc` mit Option `utf8` zur direkten Eingabe von Umlauten und ß.

`booktabs` für typografische Tabellen, s. Abschnitt 9

`PSTricks` zur Ausgabe von PostScript-Grafik, s. Abschnitt 16.2

`graphicx` zum Einbinden von Bildern, s. Abschnitt 10

`hyperref` für anklickbare Links in elektronischen Dokumenten, s. Abschnitt 16.1

6.5. Texteingabe

Der Inhalt des Dokuments – der eigentliche Text – steht zwischen `\begin{document}` und `\end{document}`.

LaTeX wird für die meisten Dokumentklassen den Text im Blocksatz ausgeben. Mehrfache Leerzeichen werden ignoriert, ebenso Zeilenumbrüche und mehrfache Leerzeilen.

(Leerzeile)

bewirkt im Dokument einen Zeilenumbruch und Einzug (indent) der nächsten Zeile (neuer Absatz).

`\\`

bewirkt Zeilenumbruch ohne Einzug, wichtig für Tabellen.

`\usepackage[utf8]{inputenc}`

erlaubt die direkte Eingabe von Umlauten usw.

6.6. Sonderzeichen, Umlaute, Anführungszeichen

Obwohl man in LaTeX dank des Paketes `inputenc` auch Umlaute u.ä. direkt eingeben kann, wird aus Gründen der Kompatibilität empfohlen, nur ASCII-Zeichen zu verwenden. U.a. kann man mit folgenden Befehlen die entsprechenden Sonderzeichen erzeugen:

`\"a`	ä	`\'{\i}`	í	`\~{n}`	ñ	`\`	*Leerzeichen*	``` `` ```	“H
`\"o`	ö	`\'{e}`	é	`\c{c}`	ç	`\%`	%	`''`	H”
`\"u`	ü	`` \`{a} ``	à	`\v{c}`	č	`\&`	&	`"` `` ` ``	„H
`\ss{}`	ß	`\^{o}`	ô	`\AA`	Å	`\#`	#	`"'`	H“

7. Gliederung

7.1. Gliederungsbefehle

Überschriften für die Abschnitte eines Dokuments können manuell durch Änderung der Schrift vorgenommen werden (s. Abschnitt 13) oder komfortabeler durch Gliederungsbefehle. Beim Verwenden der folgenden Gliederungsbefehle wird LaTeX automatisch eine Zählung vornehmen und Einträge für das Inhaltsverzeichnis anlegen:

- `\part{ }`
- `\chapter{ }` (nicht in scrartcl)
- `\section{ }`
- `\subsection{ }`
- `\subsubsection{ }`
- `\paragraph{ }`
- `\subparagraph{ }`
- `\subsubparagraph{ }`

Beispiel: `\section{Einleitung}`
Als optionales Argument (mit [ ]) kann der Titel angegeben werden, der im Inhaltsverzeichnis bzw. für Kolumnentitel verwendet wird.
Beispiel: `\section{Theoretische Grundlagen}[Theorie]`

7.2. Inhaltsverzeichnis

\tableofcontents

Einfügen des Inhaltsverzeichnisses an der Stelle im Dokument, an der dieser Befehl steht, i.d.R. kurz nach `\begin{document}`.

7.3. Titelei

Es wird empfohlen, die Titelseite gesondert zu setzen und dem Dokument später hinzuzufügen (z.B. mittels `pdftk`. Dennoch kann auch mit LaTeX eine einfach Titelseite erstellt werden.

\title{ }

Angabe eines Titels für das Dokument.

\author{ }

Der oder die Autor(en).

\date{ }

Manipulation des Datums, das für die Titelseite benutzt wird. Default ist das aktuelle Datum, das auch mit `\today` abgerufen werden kann.

\maketitle

Erstellt aus `title`, `author` und `date` einen entsprechend formatierten Titel, praktisch z.B. für Vorträge (`\documentclass{beamer}`).

8. Umgebungen, Listen

Text, der speziell formatiert werden soll, z.B. Tabellen, befindet sich i.d.R. in einer *Umgebung* (environment), die mit

```
\begin{umgebung}
 ...
\end{umgebung}
```

umschlossen wird. Umgebungen werden vom übrigen Text abgesetzt, meist mit einem zusätzlichen vertikalen Abstand.

8.1. Listen

Listen lassen sich (begrenzt) ineinander schachteln.

itemize

Stichpunktliste. Die einzelnen Stichpunkte werden mittels **\item** aufgeführt.

Beispiel:

```
\begin{itemize}
 \item Amsel
 \item Drossel
\end{itemize}
```

Resultat:

- Amsel
- Drossel

enumerate

Aufzählung, analog zu `itemize`.

Beispiel:

```
\begin{enumerate}
 \item Ende
 \item Anfang
\end{enumerate}
```

Resultat:

1. Ende
2. Anfang

Wird in einer Aufzählung eine weitere Aufzählung begonnen, so werden für die Nummerierungshierarchie folgende Zeichen verwendet:

1. Arabische Ziffern mit Punkt: 1.
2. Kleinbuchstaben mit runder Klammer: a)
3. Kleine römische Ziffern mit Punkt: i.
4. Großbuchstaben mit Punkt: A.

8.2. Weitere Umgebungen

center

Zentrierter Textsatz, z.B. für Tabellen, Abbildungen.

Beispiel:

```
\begin{center}
Zentieren ist\\
einfach
\end{center}
```

Resultat:

Zentrieren ist
einfach

verbatim

Wortwörtliche Ein- und Ausgabe von Text inklusive \, { } usw., deren besondere Bedeutung ignoriert wird. Der Text wird in einer Type-Writer-Schriftart (Monospace-Font) dargestellt und Leerzeichen und Zeilenumbrüche werden beachtet. Nützlich für Quelltext.

Beispiel:

```
\begin{verbatim}
 #include <iostream>
 using namespace ::std;
 int main() { return 0 ;}
\end{verbatim}
```

Resultat:

```
#include <iostream>
using namespace ::std ;
int main () { return 0 ;}
```

Hinweis: Mittels des Befehls `\verb+ ... +` kann ein ähnlicher Effekt auch im laufenden Text erreicht werden. Dabei dient das erste Zeichen nach `verb` (hier: +) als Begrenzer bis zu dem beim nächsten Erscheinen `verb` gilt.

alltt

Ähnlich wie `verbatim`, allerdings behalten \ und { } ihre Bedeutungen, daher kann z.B. `\em` usw. benutzt werden.
Mathematische Formeln können in `\( \)` gesetzt werden, für Super- bzw. Subskripte muss dann `\sp{ }` bzw. `\sb{ }` verwendet werden.

9. Tabellen

Die verbreiteste Tabellen-Umgebung ist `tabular`, es gibt aber für jeden erdenklichen Zweck (z.B. quer, bunt) spezielle Tabellen-Umgebungen.

tabular

Die Umgebung `tabular` benötigt ein Pflichtargument in { }, dieses beschreibt Anzahl der Spalten und einer Tabelle und die Ausrichtung der Spalteninhalte.

Beispiel:

```
\begin{tabular}{lcr}
 Stadt    & Land  & Fluss \\
 Potsdam & Polen & Po
\end{tabular}
```

Resultat:

Stadt	Land	Fluss
Potsdam	Polen	Po

Folgende Spaltenargumente sind ohne Zusatzpakete verfügbar:

- **`l`** linksbündig
- **`c`** centered (zentriert)
- **`r`** rechtsbündig
- **`|`** vertikale Linie (zw. zwei Spalten)
- **`p{`*`width`*`}`** Spalte mit fester Breite *width*

Beispiel: `\begin{tabular}{l|cc|p{1cm}|r}`

table

Um mit `tabular` erzeugte Tabellen vom Text abzusetzen, mit einer Überschrift auszustatten etc. kann man diese in die `table`-Umgebung einschließen. Table-Umgebungen sind *Gleitobjekte* (floating objects), die, da sie den Satzspiegel unterbrechen, von LaTeX an eine passende Stelle verschoben werden. Durch Angabe optionaler Argumente kann dies beinflusst werden:
`\begin{table}[htbp!]`

- **`h`** hier, falls möglich
- **`t`** top (oben auf der Seite)
- **`b`** bottom (unte auf der Seite)
- **`p`** page (auf extra Seite sammeln)
- **`!`** Überschreiben bestimmter Beschränkungen
- **`H`** Hier! (unbedingt), mittels Paket `float`

Die Voreinstellung ist `[tbp]`. Die Option `H` bewirkt leider oft, dass zwischen Tabelle und Seitenende kein Text mehr eingefügt wird.

\caption{ }

\captionabove{ }

Angabe einer Tabellenüberschrift, die Form `\captionabove{ }` funktioniert mit den KOMA-Script-Klassen und formatiert korrekt.

Es wird automatische ein Nummerierung erzeugt, diese kann mittels `\label{ }` und `\ref{ }` referenziert werden.

Man beachte, dass `\label{ }` erst nach `\caption{ }` korrekt funktioniert und innerhalb derselben Umgebung stehen muss.

\label{ *labelname* }

Anlegen eines Labels mit Namen *labelname*. Der Labelname kann dann vom Befehl `\ref{ }` als Argument referenziert werden.

\ref{ *labelname* }

Referenzieren eines Labels (mit Namen *labelname*). Im Dokument erscheint dann anstelle von *labelname* eine entsprechende Nummer einer:

- Gleichung
- Abbildung oder Tabelle
- Section, Subsection, etc.
- Aufzählung

\hline

Ausgabe einer horizontalen Linie zwischen zwei Zeilen, kann nur nach einem Zeilenumbruch `\\` stehen.

\toprule

\midrule

\bottomrule

Erfordert das Einbinden des Pakets `booktabs`. Ähnlich zu `\hline`, werden horizontale Linien, allerdings verschiedener Dicke und mit korrekten vertikalen Abständen, gezeichnet.

Linientyp	Ort
`\toprule`	oben, vor Tabellenkopf
`\midrule`	nach Tabellenkopf, im Tabellenkörper
`\bottomrule`	unten, nach letzter Zeile

10. Abbildungen/Grafiken einbinden

Folgende Pakete sind gebräuchlich, um Grafikdateien in das Dokument einzubinden:

`graphicx`, `graphics`, `epsf`

wobei das erste empfohlen wird, da es am flexibelsten ist. Das Paket `graphicx` unterstützt zusammen mit `pdflatex` folgende Grafikformate: `pdf`, `png`, `jpeg`

Zusammen mit `latex` werden die PostScript-Formate `eps` und `ps` unterstützt.

\includegraphics[ ]{*bild.pdf*}

Erfordert: `\usepackage{graphicx}`

Einbinden einer Bilddatei *bild.pdf* an der Stelle, an der der Befehl steht. In den meisten Dokumenten empfiehlt es sich, die Abbildung innerhalb einer `figure`-Umgebung einzubinden. In den optionalen eckigen Klammern können Angaben zur beabsichtigten Größe des Bildes stehen, z.B.

`\includegraphics[angle=270,width=\textwidth,height=2cm]{fig1.jpeg}`

dreht das Bild aus `fig1.jpeg` um 270 Grad und dehnt es auf Texbreite und eine Höhe von 2 cm.

figure

Gleitobjektumgebung für Abbildungen. Ermöglicht das Setzen von referenzierbaren Bildunterschriften mittels `\caption{ }`. Die `figure`-Umgebung hat dieselben Gleitobjekteigenschaften wie `table`, d.h. mit den entsprechenden Optionen kann die Positionierung beeinflusst werden. **Beispiel:**

```
\begin{figure}[!htbp]
 \begin{center}
  \includegraphics[width=0.5\textwidth]{fig2.pdf}
  \caption{Schaltbild eines Mantelstromfilters}
  \label{fig:filter}
 \end{center}
\end{figure}
```

\listoffigures

erzeugt eine Liste der Abbildungen mit Seitenzahl, analog zum Inhaltsverzeichnis.

11. LaTeX und DVI/PostScript

Das ursprüngliche LaTeX produzierte als Ausgabe nur DVI-Dateien, die sich zu PS- oder PDF-Dateien umwandeln lassen. DVI-Dateien (*de*vice *i*ndependent) lassen sich mittels spezieller DVI-Viewer (z.B. Yap für Windows, oder Xdvi für Linux) anzeigen. DVI-Dateien enthalten noch keine Abbildungen, sondern nur Verweise auf die entsprechenden Dateien, die je nach DVI-Viewer mitangezeigt werden.

Einige Dokumentklassen (z.B. `aa`) sowie das Paket `PSTricks` funktionieren nur mit `latex` und DVI-Output.
`latex` unterstützt nur PostScript-Dateien beim Einbinden von Abbildungen.

Zum Aufruf von `latex` sei auf den Abschnitt 6.1 verwiesen.

	`latex`	`pdflatex`
Ausgabe	`dvi`	`pdf`
PS-Funktionen	ja!	nein
Grafikinput	eps, ps	jpeg, pdf, png
Microtyping	beschränkt	Paket `microtype`

12. Dokumentklassen

Die Standardklassen von LaTeX sind für amerikanische Papierformate ausgelegt. Die Größe des Satzspiegels lässt sich ggf. mithilfe des Pakets `typearea` beeinflussen.

Komfortabler ist allerdings der Gebrauch der sog. KOMA-Script-Klassen, die in jeder TeX-Installation enthalten sind. Für jede der Standardklassen gibt es eine entsprechende KOMA-Scriptklasse, die DIN-Papiergrößen benutzt und intern `typearea` zur Satzspiegelkonstruktion verwendet.

Die Größe des Satzspiegels, sprich die Breite der Ränder, wird am einfachsten mittels des `DIV`-Wertes bestimmt. Bei den Klassen `scrartcl`, `scrreprt` und `scrbook` ist ein `DIV=10` bei einer Schriftgröße von `11pt` für DIN A4 voreingestellt - die Standarklassen von LaTeX verwenden übrigens `10pt` als Standardschriftgröße. Ein größerer DIV-Wert bewirkt einen größeren Satzspiegel, also schmalere Ränder, z.B.:

```
\documentclass[twosided,DIV=15,BCOR=9mm]{scrartcl}
```

Standardklasse	KOMA-Script-Klasse	
article	scrartcl	
report	scrreprt	
book	scrbook	(mit Gliederungsbefehl `\chapter{ }`)
letter	scrlttr2	(für Briefe)

bewirkt ein zweiseitiges Layout und berücksichtigt bei der Satzspiegelberechnung eine Korrektur für die Bindung, welche 9 mm des Seitenrandes wegnimmt.

12.1. Dokumentklassen für spezielle Dokumente

Die Klasse `sciposter` erlaubt die Verwendung von Papierformaten bis DIN A0 und passt die Schriftgröße sinnvoll an.

Mittels der Klasse `beamer` lassen sich elegante Vorträge für das Gespann Computer/Projektor erstellen.

Wissenschaftlich Journale definieren häufig ihre eigenen (häufig leider nicht fehlerfreien) Dokumentklassen, z.B. `aa`, `mnras`.

13. Schriftgröße und Schriftart

Die voreingestellte Schriftgröße der Dokumentklasse (z.B. 11 pt bei KOMA-Script) lässt sich i.d.R. global und lokal ändern:

```
\documentclass[12pt]{scrartcl}
```

ändert die Schriftgröße des normalen Textes auf 12 pt. Darüberhinaus sind auch lokale, meist relative Schriftgrößenänderungen möglich, z.B.

```
{\Large{}Eine \"Uberschrift}\\
Normaler Text
```

ergibt

Eine Überschrift
Normaler Text

Es gibt folgende Größenangaben, die die Schrift i.d.R. um jeweils 1 pt ändern:

```
\tiny
\scriptsize
\footnotesize
\small
\normalsize
\large
\Large
\LARGE
\Huge
```

13.1. Schriftform

Zu jedem Font (Schriftart) gibt es mehrere Formen, die durch entsprechende Befehle (s. Tab. 1) ausgewählt werden können. Diese Wechsel gelten innerhalb einer Umgebung oder bis zum nächsten Wechsel. Man kann die Wirkung aber auch durch Einklammern mit geschweiften Klammern `{ }` auf einen Bereich beschränken, gleiches gilt für den Wechsel der Schriftgröße. Möchte man bestimmte Formen kombinieren, so funktioniert dies meist über den Wechsel der Schriftfamilie:

```
{\bf\sffamily{}Fett ohne Serifen}
```

Fett ohne Serifen

Tabelle 1: Befehle zum Wechsel der Schriftform

`\sf`	Sans Serif – ohne Serifen
`\em`	*emphasize* – meist durch kursive, geneigte Schrift (=`\it`)
`\sl`	*slanted* – geneigte Schrift
`\tt`	`TypeWriter` – Monospace-Schrift mit breiten Serifen
`\rm`	Roman – normale Schrift mit Serifen
`\sc`	SMALL CAPITALS – Kapitälchen
`\bf`	**Bold Face** – Fettdruck

14. Formeln setzen

Sicherlich einer der wichtigsten Gründe LaTeX zu verwenden ist der perfekte Formelsatz.

$ $

Formeln im laufenden Text setzen, z.B. `$\sum \frac{1}{x^2}$` ergibt $\sum \frac{1}{x^2}$, die Formel wird weder nummeriert noch besonders abgesetz. Die Größe ist auf `\textstyle` voreingestellt.

\(\)

Formel im laufenden Text setzen, z.B. `\(c\neq\sqrt[3]{a^3+b^3}\)` erzeugt $c \neq \sqrt[3]{a^3+b^3}$, funktioniert auch in der `alltt`-Umgebung.

$$ $$

vom Text abgesetzte Formel ohne Nummerierung erzeugen.**Beispiel:**

```
$$\int_V\nabla\cdot\vec{F}d^{(n)}V
   = \oint_S\vec{F}\cdot\vec{n}\,d^{(n-1)}S $$
```

$$\int_V \nabla \cdot \vec{F} d^{(n)} V = \oint_S \vec{F} \cdot \vec{n}\, d^{(n-1)} S$$

eqnarray, eqnarray*

Umgebung für abgesetzte, mehrzeilige Formel mit fortlaufender Nummerierung für jede Zeile. Textgröße ist auf `\displaystyle` voreingestellt. Intern funktioniert die `eqnarray`-Umgebung wie eine Tabelle mit `{rcl}`. Die Sternform unterdrückt die Nummerierung, welche man ebenfalls zeilenweise mit `\nonumber` ausschalten kann. **Beispiel:**

```
\begin{eqnarray}
\zeta (2) & = & \sum_{n=1} ^\infty \frac{1}{n^2}\\
          & = & \frac{\pi}{6} \label{eq:zeta2}
\end{eqnarray}
```

$$\begin{aligned} \zeta(2) &= \sum_{n=1}^{\infty} \frac{1}{n^2} && (1) \\ &= \frac{\pi}{6} && (2) \end{aligned}$$

Zeilen einer Formel, die mit einem `\label{}` versehen wurden, können im Text einfach referenziert werden, z.B. wurde die Gl. (2) mittes `Gl.~(\ref{eq:zeta2})` referenziert.

equation, **equation***

wie `eqnarray`, allerdings nur für einzeilige Formeln.

\frac { }{ }

Darstellung von Brüchen, auch verkettet, z.B. `\frac{1}{1+\frac{1}{n}}` für $\frac{1}{1+\frac{1}{n}}$.

\sum, **\int**

Darstellung des Summen- bzw. Integralzeichens, z.B.:
`\int_{1/m}^1 \sup_n f_n(x) dx = \sum_{n=1}^{m-1} \frac{1}{n+1}`

$$\int_{1/m}^{1} \sup_n f_n(x)dx = \sum_{n=1}^{m-1} \frac{1}{n+1}$$

_ , **^**

Das dem Unterstrich folgende Zeichen wird tiefergestellt, mehrere Zeichen müssen mit geschweiften Klammern zusammengehalten werden. Das Caret ^ bewirkt analog ein Hochstellen, siehe Beispiel für `\sum`.

15. Bibliografie

Eine automatische Bibliografie kann auf zwei Arten erstellt werden: mittels der `bibliography`-Umgebung oder mittels des Programms `bibtex`.

15.1. bibtex

Für `bibtex` werden die Referenzen in einer separaten Datei gesammelt, die die Dateiendung `.bib` trägt. Ein typischer Eintrag hat folgendes Format:

```
@ARTICLE{acknein2003,
   author = {{Acker}, A. and {Neiner}, C.},
    title = "{Quantitative classification of WR nuclei
             of planetary nebulae}",
  journal = {A \& A},
     year = 2003,
   volume = 403,
    pages = {659-673}
}
```

In der ersten Zeile wird die Art der Referenz hinter einem `@` angegeben, z.B. `ARTICLE`, `BOOK`, `INPROCEEDINGS`, `THESIS`, danach wird ein interner Name (*clabel*) für das Referenzieren im Dokument angegeben (ähnlich einem `label`-Namen). Dieser Name sollte eine vierstellige Jahreszahl enthalten. Die angegebenen geschweiften Klammern sind Pflicht. Die Daten des Eintrages, z.B. `author`, `year`, werden jeweils mit einem = zugeordnet und mit einem Komma abgetrennt.

\cite {*clabel*}

Referenzieren eines Bibliografieeintrages mit internem Namen *clabel*.

\bibliographystyle {*style*}

Obligatorische Angabe des Bibliografiestils in der Präambel. Typische Stile sind z.B. `plain`, `aa` usw.

\bibliography {*datei*}

Einfügen der Bibliografie aus der Datei *datei*`.bib` – ohne Angabe der Endung `.bib`.

bibtex *texdatei.aux*

Aufruf des Programms `bibtex` auf der Kommandozeile zum Erzeugen der Datei *bibdatei.bln* aus *bibdatei.bib*. Dafür wird die `texdatei.aux` des TEX-Dokuments ausgewertet und nur diejenigen Referenzen verarbeitet, für die ein `cite`-Befehl im TEX-Dokument steht.

Das Paket `natbib` erweitert den `\cite{ }`-Befehl u.a. um

\citep [*davor*][*danach*] {*clabel*}

Referenzieren eines Bibliografieeintrages, wobei um das Zitat runde Klammern gesetzt werden. In diese runde Klammern kann vor dem Zitat mittels des ersten optionalen Arguments oder nach dem Zitat durch das zweite optionale Argument weiterer Text eingefügt werden. **Beispiel:**

```
\citep[siehe auch][und Referenzen darin]{acknein2003}
```

ergibt (siehe auch Acker & Neiner 2003, und Referenzen darin)

15.2. thebibliography

Alternativ zu `bibtex` kann auch eine *dokumentinterne* Bibliografie mithilfe der `thebibliography`-Umgebung angelegt werden:

thebibliography

Umgebung zur Definition von `\bibitem`s, diese werden darin angezeigt und lassen sich im Text mittels `\cite{ }` zitieren.

Die `thebibliography`-Umgebung hat ein Pflichtargument, das benutzt wird, um den Einzug der Bibitems festzulegen, der Einzug ist so breit, wie das Pflichtargument (beliebige Zeichen) lang ist.

\bibitem [*marke*] {*clabel*} *Quelle*

Anlegen eines eines Bibliografieeintrages. Die Markierung [*marke*] ist optional und legt die Darstellung im Text für das Zitat fest, voreingestellt ist eine fortlaufende Nummerierung in eckigen Klammern, z.B. [1] für das erste Bibitem. Das Pflichtargument *clabel* definiert den interen Namen, der dann mittels `\cite{`*label*`}` verwendet werden kann. Nachfolgend ist die Angabe der Quelle, die selbst formatiert werden muss.

\cite {*clabel*}

Referenzieren eines Bibliografieeintrages mit Namen *clabel*.

Beispiel

```
\begin{thebibliography}{einzugstiefe}
\bibitem[Acker \& Neiner 2003]{acknein2003} Acker, A. \& Neiner, C.,
        {\it A\&A}, 2003, {\bf 403}, p.\ 659
\end{thebibliography}
```

ergibt

[Acker & Neiner 2003] Acker, A. & Neiner, C., *A&A*, 2003, **403**, p. 659

16. Weitere nützliche Pakete

16.1. hyperref

Das Paket `hyperref` erstellt für alle Referenzen, Zitationen und URLs anklickbare Links in einem PDF-Dokument, d.h. es kann damit innerhalb des Dokuments navigiert werden oder im Falle einer URL oder E-Mail-Adresse die entsprechende Anwendung gestartet werden.

Das Paket `hyperref` muss dafür stetst als letztes Paket eingebunden werden. Nur zusammen mit `pdflatex` werden so erzeugte Hyperlinks im Dokument auch korrekt umgebrochen. **Beispiel:** Einstellungen als Paketoptionen:

```
\usepackage[hyperindex=true % anklickbarer Index
            ,colorlinks=true % false fuers Drucken
            ,linkcolor=blue % Farbe der Links (default: rot)
            ,citecolor=darkblue, % Farbe der Zitate (default: gruen)
            breaklinks=true % Umbrechen von Links erlauben (pdflatex)
           ]{hyperref}
```

16.2. PSTricks

Eine äußerst mächtige Sammlung von Paketen zum Erzeugen von PostScript-Vektor-Grafiken existiert unter dem Namen `PSTricks`, z.B.

```
\usepackage{pstricks} % grundlegende Funktionen wie pspicture und psline
\usepackage{pst-all} % Einbinden aller PSTricks-Pakete - nicht empfohlen
\usepackage{pst-eps} % erzeugen einer korrekten BoundingBox fuer eps
```

Um PSTricks verwenden zu können *muss* `latex + dvips` benutzt werden, siehe Beispiel zu `multido`.

16.2.1. multido

Dieses Paket wird häufig zusammen mit PSTricks eingesetzt und ermöglicht einfache Schleifen in LaTeX. **Beispiel:**

```
 ...
\usepackage{pstricks}
\usepackage{multido}
\usepackage{calc}
\begin{document}
\newlength{\xdist}
\begin{pspicture}(-2,1)(14,-14.5)
\multido{\i= 2 + 1 }{8}{
 \setlength{\xdist}{1.0cm * \real{\i}}
 \psline{<->}(\xdist,-13.6)(\xdist,-2.)
}
\end{pspicture}
\end{document}
```

Teil IV.
Index

Index