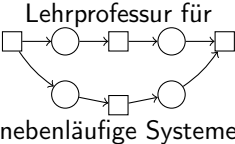


Vorkurs Informatik
Organisation / Ablauf / Vorbereitungen

Robert Lorenz

Tag 1



Der Vorkurs ist für Studienanfänger aller Bachelor-Studiengänge mit Haupt-, Teil- oder Nebenfach Informatik:

- Informatik, Geoinformatik, Wirtschaftsinformatik, Ingenieurinformatik, Medizinische Informatik
- Data Science, Mathematik und Informatik
- Mathematik, Physik und Geographie mit Nebenfach Informatik
- Wirtschaftsmathematik, Ingenieurwissenschaften

Verwaltungssystem des Lehrstuhls

- Anmeldung zum Vorkurs
- Rundmails und aktuelle Nachrichten / Ankündigungen
- Informationen zum Vorkurs
- Lehrmaterialien (Folien, Lehrvideos, Übungsaufgaben, Anleitungen)

Zugang

<https://digicampus.uni-augsburg.de>

- Thema 1:
Ablauf und Organisation, Installation und Benutzung der benötigten Software (Editor, gcc-Compiler, Shell / Kommandozeile)
- Thema 2:
Variablen, Konstanten, Wertzuweisungen, Rechenausdrücke, Ausgaben auf Kommandozeile (`printf`), der Datentyp `int`, Generierung von Zufallszahlen (`srand`, `rand`)
- Thema 3:
Fallunterscheidungen (`if-else`), Logische Ausdrücke, der Datentyp `char`, ASCII, Benutzung der Standard-Bibliothek (`ctype.h`)

- Thema 4:
Wiederholungen (`while`, `for`), Typumwandlungen,
Rundungen, der Datentyp `double`, Benutzung der
Standard-Bibliothek (`math.h`)
- Thema 5:
Funktionen
- Thema 6:
Arrays und Strings

Dazu solltest du alle Lehrmaterialien zum

Thema 1 (ca. 45 Minuten Lehrvideos)

im Selbststudium zu Hause durcharbeiten:

- Zuerst die Übersicht zu den Lehrmaterialien durchlesen und Änderungen seit der Erstellung der Lehrvideos beachten.
- Dann die Videos ansehen (Mitschreiben!), und ergänzend zur eigenen Mitschrift die Folien durchgehen.
- Wer einen eigenen PC / Laptop hat, installiert bitte alle benötigten Programme vorab (siehe Anleitungen)
- Bei Problemen meldet euch bei einem Betreuer (Kontakte siehe Digicampus)

- Zu jedem der Themen stehen Lehrvideos, Folien und Übungsaufgaben zur Verfügung, die du dir jeweils im Selbststudium erarbeitest.
- Bei Fragen und Problemen wirst du von Tutoren betreut
- *Hinweis zum Selbststudium: Zuerst die Übersicht zu den Lehrmaterialien durchlesen, dann die Videos ansehen (Mitschreiben!), und ergänzend zur eigenen Mitschrift die Folien durchgehen.*

Zugang zur Betreuung

Betreuung gibt es vor Ort in Präsenz: Räume siehe Digicampus

Der erste Tag (Montag)

- 08:00 - 12:00 Uhr: Selbststudium der Folien und Lehrvideos
- 13:00 - 14:00 Uhr: Begrüßung in Präsenz (Raum: siehe Digicampus)
- 14:00 - 17:15 Uhr: Betreute Bearbeitung der Übungsaufgaben

Tagesablauf Dienstag - Donnerstag

- 8:00 - 12:00 Uhr: Selbststudium der Folien und Lehrvideos
- 14:00 - 17:15 Uhr: Betreute Bearbeitung der Übungsaufgaben

- Es gibt keine Zuordnung der Themen zu einzelnen Tagen
- Jeder bearbeitet die Themen in seinem eigenen Tempo

Aufwandsübersicht

- Thema 2: ca. 95 min Lehrvideos, 90 min Lesezeit, 90 min Programmierzeit
- Thema 3: ca. 79 Minuten Lehrvideos, 90 min Lesezeit, 120 min Programmierzeit
- Thema 4: ca. 107 Minuten Lehrvideos, 90 min Lesezeit, 150 min Programmierzeit
- Thema 5: ca. 59 Minuten Lehrvideos, 90 min Lesezeit, 180 min Programmierzeit
- Thema 6: ca. 77 Minuten Lehrvideos, 120 min Lesezeit, 180 min Programmierzeit

Du hast am Vorkurs teilgenommen, hast aber noch Fragen?

Du hast den Vorkurs verpasst (weil du dich zum Beispiel erst danach immatrikuliert hast)?

Dann melde dich einfach nachträglich im Digicampus zum Vorkurs an:

- Die Lehrmaterialien erarbeitest du dir im Selbststudium
- Bei Fragen meldest du dich bei einem Betreuer - dieser vereinbart mit dir einen Termin (online oder auch in Präsenz)

Im Vorkurs sollst du lernen, selbständig, zügig und ohne Benutzung von Hilfsmitteln einfache kleine Programme in C schreiben und ausführen. Das ist eine Grundvoraussetzung für den Beginn des Informatik-Studiums.

Das machst du nach dem Vorkurs

- Weiter regelmäßig Programme im Selbststudium schreiben
- Man benötigt mindestens 100 Stunden eigenständiges Programmieren, um am Ende des ersten Semesters die Klausur *Informatik 1* zu bestehen.

- Der Programmierer schreibt den **Quellcode** eines Programms. Der Quellcode ist eine einfache **Textdatei**.
- Der Quellcode wird durch einen **Compiler** in **Maschinencode** übersetzt, den der Rechner ausführen kann.

Das braucht man also zum Programmieren

- Einen einfachen **Texteditor** (Empfehlungen später), um den Quellcode zu erstellen.
- Eine **Bibliothek** von vorinstallierten Funktionen (z.B. zur Ein- und Ausgabe), die man in seinem Programm benutzen kann
- Einen **Compiler**, mit dem man den Quellcode in Maschinencode übersetzen kann

GNU Compiler Collection (gcc)

- Bibliotheken und Compiler für die Programmiersprache C (verfügbar für Windows, Linux und Mac)
- Installationshilfe: Anleitungen im Digicampus

Empfohlene Texteditoren für den Vorkurs

- Notepad++ (Windows):
<https://notepad-plus-plus.org/>
- Visual Studio Code (Windows, Mac OS, Linux):
<https://code.visualstudio.com>

C Programmieren von Anfang an (H. Erlenkötter, rororo)

- Sehr leichte Einführung; geht pädagogisch vor
- Führt in vielen Themen nur in Grundlagen ein
- Vermittelt kaum vertieftes Wissen
- *Ausreichend für den Vorkurs*

Programmieren in C (Kernighan / Ritchie, Hanser)

- Das Standard- und Nachschlagewerk für C
- Inhaltlicher Aufbau/Beispiele anspruchsvoll
- *Geeignet als Ergänzung der Vorlesung Informatik 1*

C von A bis Z (J. Wolf, Rheinwerk Computing)

- Online verfügbar unter http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/
- Ideal zum Nachschlagen
- Themen werden kompakt zusammengefasst
- Zu Beginn muss man mit einigen unbekannten Begriffen zurecht kommen, die erst später genau beschrieben werden

Wikibooks C-Programmierung

- Online verfügbar unter
`https://de.wikibooks.org/wiki/C-Programmierung`
- Einzelne Themen kurz und schnell erklärt

Grund-Wissen: Grundkurs in C

- Online verfügbar unter
`https://www.grund-wissen.de/informatik/c/index.html`
- Alle Befehle kurz und knapp erklärt

C-Programme

- Ein C-Programm ist eine ungeordnete Sammlung von Funktionsdefinitionen (d.h. die Reihenfolge der Definitionen ist nicht wichtig).
- Es muss immer genau eine Hauptfunktion `main` geben; Wird das Programm gestartet, so werden die Anweisungen dieser Hauptfunktion ausgeführt.
- Meistens werden auch noch sog. Standard-Bibliotheken mit vordefinierten Funktionen eingebunden
Beispiel: Aufruf der Funktion `printf` für Ausgaben in der Kommandozeile

Programmstruktur

Beispiel

```
1  #include <stdio.h>
2
3  int ggT(int m, int n) {
4      int a = m;
5      int b = n;
6      int r;
7      while (b > 0) {
8          r = a % b;
9          a = b;
10         b = r;
11     }
12     return a;
13 }
14
15 int main(void) {
16     int erg = ggT(30, 42);
17     printf("%i\n", erg);
18     return 0;
19 }
```

Einbindung der Bibliothek `stdio.h`

Definition der `ggT`-Funktion mit zwei Eingaben

Definition der `main`-Funktion
Aufruf der `ggT`-Funktion für 30 und 42
Aufruf der `printf`-Funktion

Programmstruktur

Beispiel

```
1  #include <stdio.h>
2
3  int ggT(int m, int n) {
4      int a = m;
5      int b = n;
6      int r;
7      while (b > 0) {
8          r = a % b;
9          a = b;
10         b = r;
11     }
12     return a;
13 }
14
15 int main(void) {
16     int erg = ggT(30, 42);
17     printf("%i\n", erg);
18     return 0;
19 }
```

- Wo steht überhaupt dieser Text?
- Wie bringt man den Rechner dazu, ihn auszuführen?
- Wie sieht es mit wechselnden Werten für `m, n` aus?

Ein C-Programm wird einfach als Textdatei erstellt:

- Der Programmtext heißt **Quellcode**.
- Für den Quellcode benutzen wir ausschließlich den **ASCII-Zeichensatz** (Details später, Achtung: dieser Zeichensatz enthält keine Umlaute und kein scharfes ß!)
- Die Textdatei wird mit der Endung `.c` abgespeichert (nicht `.txt`!).
- Der Quellcode ist noch kein durch den Rechner ausführbares Programm

Syntax

- Die **Syntax** einer Programmiersprache legt fest, wie der Quellcode aufgebaut werden muss, damit der Rechner ihn verstehen und ausführen kann (die Syntax entspricht also den Grammatikregeln für den Quellcode).
- Zur Syntax gehört, welche Befehle und Bibliotheken mit vorinstallierten Funktionen zur Verfügung stehen, und wie diese aufgerufen werden können.
- Die Syntax der Programmiersprache C wird durch verschiedene aufeinander aufbauende **Programmier-Standards** festgelegt.

Programmier-Standards

- Für die Programmiersprache C existieren verschiedene Standards, die zur Definition und Normierung der Sprache entwickelt wurden und werden.
- Im Vorkurs (und später in der Veranstaltung Informatik 1) halten wir uns an den Standard **C11** (das ist neu - bis 2024 haben wir den Standard C89 verwendet).
- C11 baut auf den Standards C89 (ISO C90, ANSI C) und C99 auf und bringt ein paar praktische Erweiterungen und Vereinfachungen mit
- Für den Vorkurs ist der Funktionsumfang von C89 ausreichend

Definition C89

<http://port70.net/~nsz/c/c89/c89-draft.html>

Definition C99

<https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf>

Definition C11

<https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf>

- Sind nicht notwendig für den Vorkurs
- Nicht zum Lernen gedacht, sondern zum Nachschlagen

Dokumentation C89

<https://www.grund-wissen.de/informatik/c/standardbibliothek.html>

Deutsche Online-Dokumentation (knappe Übersicht; *Ausreichend für den Vorkurs*)

Dokumentation C89 - C23

<https://devdocs.io/c/>

Englische Online-Dokumentation (komplette Übersicht; knappe Beschreibungen; *nicht notwendig im Vorkurs*)

Programmier-Konventionen sind Format-Regeln zur Erstellung von strukturiertem Code, wie zum Beispiel

- Regeln der Namensgebung für Variablen und Funktionen
- Regeln für Zeilenumbrüche, Einrückungen und Leerzeichen
- ...

Programmier-Konventionen führen zu

- besserer Lesbarkeit und Wartbarkeit von Code
- geringer Fehleranfälligkeit von Code
- besserer Team- und Projektarbeit

Programmier-Konventionen

Ein gut strukturiertes Programm

```
1 #include <stdio.h>
2
3 int ggT(int m, int n) {
4     int a = m;
5     int b = n;
6     int r;
7     while (b > 0) {
8         r = a % b;
9         a = b;
10        b = r;
11    }
12    return a;
13 }
14
15 int main(void) {
16     int erg = ggT(30, 42);
17     printf("%i\n", erg);
18     return 0;
19 }
```

Das Format des Quellcodes hat keine Auswirkungen auf dessen Korrektheit.

Funktion ggT unstrukturiert

Unstrukturierter Code ist zwar korrekt, ...

```
1 #include <stdio.h>
2 int ggT (int m,int n) {int a=m;int b=n;int r;
3 while (b>0){r=a%b;a=b;b=r;}return a;} int main()
4 {int erg = ggT(30,42);printf("%i\n", erg);return 0;}
```

... aber auch schwer verstehbar

Beispiele für **Programmier-Konventionen** (oder **Coding Conventions**) sind:

<https://www.kernel.org/doc/Documentation/process/coding-style.rst>

Linux Kernel Style (Kapitel 1-4, 6, 8, 16) - nur zum Nachschlagen!

<https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html>

C Coding Standard - nur zum Nachschlagen!

Achtung

Im Vorkurs werden geeignete Konventionen implizit verwendet, aber nicht explizit besprochen.

Der Quellcode muss erst noch in (durch den Rechner) ausführbaren Code, sog. **Maschinencode**, übersetzt werden



Maschinencode

Maschinencode ist direkt durch den Rechner ausführbarer Code.

Compiler

Ein **Compiler** ist ein Programm, das Quellcode in Maschinencode übersetzt.

Dazu überprüft der Compiler den Quellcode zuvor auf Korrektheit (Übersetzbarkeit, korrekte Syntax) und liefert ggf. Fehlermeldungen.

Kompilierbarkeit

Ein Programm nennt man **kompilierbar**, wenn der Compiler keine Fehlermeldungen liefert.

Wir verwenden im Vorkurs den gcc-Compiler. Dieser muss abhängig vom Betriebssystem auf dem Rechner ggf. noch installiert werden

Empfehlung für Windows

- Installiere MinGW ("Minimalist GNU for Windows"): enthält gcc-Compiler
- Siehe Anleitung im Digicampus

Empfehlung für MAC

- Ist ggf. schon installiert
- Kann leicht getestet und nachinstalliert werden
- Siehe Anleitung im Digicampus

Empfehlung für Linux/Unix

- gcc-Compiler ist Teil der Installationspakete

Wir rufen den gcc-Compiler über die sog. **Kommandozeile** (Details dazu später) auf.

gcc-Aufruf - einfache Version

```
gcc <Quellcode> -o <Programm>
```

gcc	Aufruf des gcc-Programms
<Quellcode>	Dateiname des Quellcodes
<Programm>	Dateiname des Programms

*Anmerkung: Die eckigen Klammern < und > **nicht mittippen** - sie kennzeichnen Platzhalter für freie Eingaben*

Schicken wir den Befehl

```
gcc bsp01.c -o prg01
```

in die Kommandozeile, so wird die Quellcode-Datei `bsp01.c` in die Programmdatei `prg01` übersetzt (d.h. in `prg01` steht der Maschinencode).

Schicken wir nun den Befehl

```
prg01
```

in die Kommandozeile, so wird das Programm ausgeführt (diesen Befehl nennen wir den Programmaufruf).

Dateinamen werden

- entweder **relativ** zum **Programmverzeichnis**
- oder **absolut** mit **Pfadangabe** beginnend beim **Wurzelverzeichnis**

angegeben (Unterschiede zwischen Betriebssystemen:
Eigenrecherche)

Programmverzeichnis

Das Verzeichnis, in dem das Programm gespeichert ist.

Beispiele für Pfadangaben

- `bsp01.c`: Relativer Dateiname, `bsp01.c` befindet sich im Programmverzeichnis.
- `D:\Beispiele\bsp01.c`: Absoluter Dateiname in Windows, `bsp01.c` befindet sich im Ordner `Beispiele` auf dem Laufwerk `D`.

- Das Programm gcc kann mit verschiedenen optionalen sog. **Kommandozeilenparametern** aufgerufen werden.
- Diese Kommandozeilenparameter werden jeweils durch ein Leerzeichen getrennt aufgeführt.
- Andere Begriffe für gcc-Kommandozeilenparameter: **gcc-Optionen**, **gcc-Compilerschalter**

Einige Kommandozeilenparameter für den Vorkurs

```
gcc <Quellcode> -o <Programm> -std:c11 -pedantic -Wall  
-Wextra
```

-std:c11	Compiler verwendet C11-Standard
-pedantic	Compiler akzeptiert ausschließlich C11
-Wall	Einige Warnungen werden angeschaltet
-Wextra	Zusätzliche Warnungen werden angeschaltet

Warnungen weisen auf unsauberen oder unstrukturierten Code hin.

- Der gcc hat viele Hundert mögliche Optionen, von denen wir nur einen Bruchteil behandeln.
- Manche sind in der Verwendung unabhängig von der Reihenfolge, manche nicht.
- Manche sind kombinierbar und manche nicht.
- Vollständige Beschreibung der gcc-Optionen: <https://gcc.gnu.org/onlinedocs/gcc/Invoking-GCC.html> (nur zum Nachschlagen)
- Hinweise bekommt man außerdem durch Benutzung der Option `-help` als Kommandozeilenparameter.

Definition (ASCII)

American Standard Code for Information Interchange

- Der ASCII-Standard nummeriert 128 wichtige Zeichen der europäisch/amerikanischen Schriften von 0 bis 127 zur Speicherung im Rechner
- Darunter sind alle lateinischen Buchstaben, aber auch **nicht druckbare** Zeichen mit ausschließlich formatierender Wirkung (z.B. Tabulatorzeichen, Zeilenumbruchzeichen)
- Einige Zeichen des deutschen Alphabets sind **nicht darunter**: z.B. ä, ö, ü, ß.

Achtung

In C-Programmen verwenden wir nur diese 128 ASCII-Zeichen

ASCII-Tabelle

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 Space		64	40	100	@ @		96	60	140	` <b>`	
1	1	001	SOH (start of heading)	33	21	041	! !		65	41	101	A A		97	61	141	a a	
2	2	002	STX (start of text)	34	22	042	" "		66	42	102	B B		98	62	142	b b	
3	3	003	ETX (end of text)	35	23	043	# #		67	43	103	C C		99	63	143	c c	
4	4	004	EOT (end of transmission)	36	24	044	$ \$		68	44	104	D D		100	64	144	d d	
5	5	005	ENQ (enquiry)	37	25	045	% %		69	45	105	E E		101	65	145	e e	
6	6	006	ACK (acknowledge)	38	26	046	& &		70	46	106	F F		102	66	146	f f	
7	7	007	BEL (bell)	39	27	047	' '		71	47	107	G G		103	67	147	g g	
8	8	010	BS (backspace)	40	28	050	((		72	48	110	H H		104	68	150	h h	
9	9	011	TAB (horizontal tab)	41	29	051	))		73	49	111	I I		105	69	151	i i	
10	A	012	LF (NL line feed, new line)	42	2A	052	* *		74	4A	112	J J		106	6A	152	j j	
11	B	013	VT (vertical tab)	43	2B	053	+ +		75	4B	113	K K		107	6B	153	k k	
12	C	014	FF (NP form feed, new page)	44	2C	054	, ,		76	4C	114	L L		108	6C	154	l l	
13	D	015	CR (carriage return)	45	2D	055	- -		77	4D	115	M M		109	6D	155	m m	
14	E	016	SO (shift out)	46	2E	056	. .		78	4E	116	N N		110	6E	156	n n	
15	F	017	SI (shift in)	47	2F	057	/ /		79	4F	117	O O		111	6F	157	o o	
16	10	020	DLE (data link escape)	48	30	060	0 0		80	50	120	P P		112	70	160	p p	
17	11	021	DC1 (device control 1)	49	31	061	1 1		81	51	121	Q Q		113	71	161	q q	
18	12	022	DC2 (device control 2)	50	32	062	2 2		82	52	122	R R		114	72	162	r r	
19	13	023	DC3 (device control 3)	51	33	063	3 3		83	53	123	S S		115	73	163	s s	
20	14	024	DC4 (device control 4)	52	34	064	4 4		84	54	124	T T		116	74	164	t t	
21	15	025	NAK (negative acknowledge)	53	35	065	5 5		85	55	125	U U		117	75	165	u u	
22	16	026	SYN (synchronous idle)	54	36	066	6 6		86	56	126	V V		118	76	166	v v	
23	17	027	ETB (end of trans. block)	55	37	067	7 7		87	57	127	W W		119	77	167	w w	
24	18	030	CAN (cancel)	56	38	070	8 8		88	58	130	X X		120	78	170	x x	
25	19	031	EM (end of medium)	57	39	071	9 9		89	59	131	Y Y		121	79	171	y y	
26	1A	032	SUB (substitute)	58	3A	072	: :		90	5A	132	Z Z		122	7A	172	z z	
27	1B	033	ESC (escape)	59	3B	073	; ;		91	5B	133	[[		123	7B	173	{ {	
28	1C	034	FS (file separator)	60	3C	074	< <		92	5C	134	\ \		124	7C	174	| 	
29	1D	035	GS (group separator)	61	3D	075	= =		93	5D	135	]]		125	7D	175	} }	
30	1E	036	RS (record separator)	62	3E	076	> >		94	5E	136	^ ^		126	7E	176	~ ~	
31	1F	037	US (unit separator)	63	3F	077	? ?		95	5F	137	_ _		127	7F	177	 DEL	

- Die Kommandozeile ist ein typischerweise textbasiertes Programm zur Eingabe von Kommandos
- Kommandos werden als Zeichenfolgen per Tastatur eingegeben und durch Drücken der Eingabetaste ausgeführt

Kommandoarten

- Navigation im Dateisystem
- Steuerung des Betriebssystems
- Ausführen von Programmen

Andere Bezeichnungen für die Kommandozeile

Befehlszeile, command-line interface / CLI, Konsole, Terminal, Shell, Bash

- Die Kommandos sind abhängig vom Betriebssystem
- Es wird jeweils das aktuelle Verzeichnis des Dateisystems angezeigt

Kommandos für Windows (kleine Auswahl) - siehe auch Anleitung im Digicampus

- `dir`: Auflistung aller Unterverzeichnisse und Dateien im aktuellen Verzeichnis
- `cd <Unterverzeichnis>`: Wechsel in ein Unterverzeichnis des aktuellen Verzeichnisses
- `cd ..`: Wechsel in das Oberverzeichnis des aktuellen Verzeichnisses (.. repräsentiert das Oberverzeichnis)
- `md <Verzeichnis>`: Ein Unterverzeichnis im aktuellen Verzeichnis erstellen
- `rd <Verzeichnis>`: Ein Unterverzeichnis im aktuellen Verzeichnis löschen
- `del <Datei>`: Eine Datei im aktuellen Verzeichnis löschen
- `<Kommando> /?`: Einen Hilfstext zur Benutzung eines Kommandos anzeigen
- `<Programm>`: Ausführung eines Programms, das sich im aktuellen Verzeichnis befindet

- Die Kommandos sind abhängig vom Betriebssystem
- Es wird jeweils das aktuelle Verzeichnis des Dateisystems angezeigt

Kommandos für MAC / Linux (kleine Auswahl)

- `ls`: Auflistung aller Unterverzeichnisse und Dateien im aktuellen Verzeichnis
- `cd <Unterverzeichnis>`: Wechsel in ein Unterverzeichnis des aktuellen Verzeichnisses
- `cd ..`: Wechsel in das Oberverzeichnis des aktuellen Verzeichnisses (.. repräsentiert das Oberverzeichnis)
- `mkdir <Verzeichnis>`: Ein Unterverzeichnis im aktuellen Verzeichnis erstellen
- `rmdir <Verzeichnis>`: Ein Unterverzeichnis im aktuellen Verzeichnis löschen
- `rm <Datei>`: Eine Datei im aktuellen Verzeichnis löschen
- `man <Kommando>`: Einen Hilfstext zur Benutzung eines Kommandos anzeigen (beenden mit q)
- `./<Programm>`: Ausführung eines Programms, das sich im aktuellen Verzeichnis befindet (. repräsentiert das aktuelle Verzeichnis)

Einige Besonderheiten / Hilfestellungen zur Ausführung von Programmen.

Kommando

Das Kommando zur Ausführung eines Programms entspricht dem Dateinamen ohne Endung

Kommandozeilenparameter

Dem Kommando können, jeweils getrennt durch ein Leerzeichen, sog. **Kommandozeilenparameter** zur Angabe von Zusatzinformationen für die Programmausführung folgen (siehe z.B. gcc-Ausführung)

Einige Besonderheiten / Hilfestellungen zur Ausführung von Programmen.

Ausführungs- und Programmverzeichnis

- **Ausführungsverzeichnis:** Aktuelles Verzeichnis bei Ausführung des Programms
- **Programmverzeichnis:** Verzeichnis, in dem sich die Programmdatei befindet
- Programm- und Ausführungsverzeichnis können verschieden sein: Damit in diesem Fall das Programm gefunden wird, muss das Programmverzeichnis in der sog. Umgebungsvariablen Path/PATH (siehe Systemeinstellungen des Betriebssystems) gelistet sein (wird bei Installation dort eingetragen oder muss nach der Installation eines Programms selbst nachgetragen werden)