Konzepte des prozeduralen Programmierens

Stand September 2022

- Prof. Dr. Oliver S. Lazar / Christian Frank

2 Start in die Programmierung

Entwicklungsumgebung, Hallo Welt!, Compiler, Kommentare,

Datentypen, Operatoren und Ausdrücke

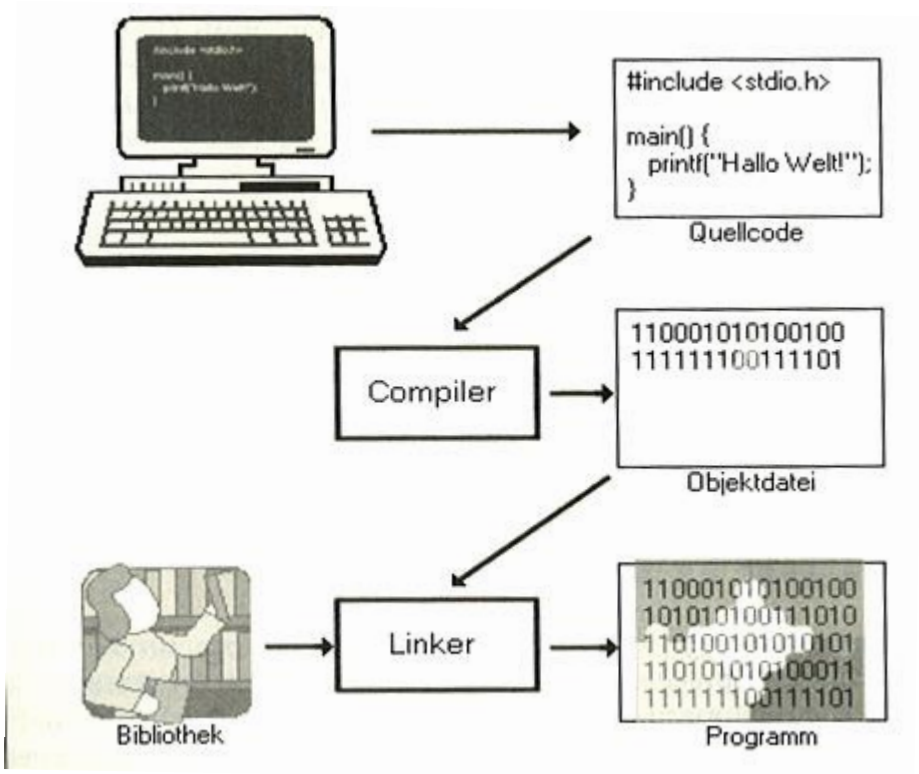
- **Texteditor**
  - C-Programme werden in einem Texteditor geschrieben
    - vi

- **Grundgerüst**

```
#include<stdio.h>
int main() {
        return 0;
}
```

Übersetzung

- Compiler übersetzt Programm in Hochsprache (z.B. C) in ein ausführbares Maschinenprogramm
- Compiler zum Selberbauen: wispy



Ablauf der Übersetzung

- Präprozessor
  - Ausführen der Präprozessoranweisungen
    - #include
    - #define
    - #if
- Compiler
  - Übertragung in Maschinencode / Objektcode
- Linker
  - Verbindung einzelner Programmmodule
  - Ausführbare Datei

Compiler

Aufgaben des Compilers:

(1) Analysephase

- Lexikalische Analyse
  - Einteilung in Schlüsselwörter, Bezeichner, Operatoren etc.

```
wert = laenge x 3,14;
```

  - Die lexikalische Analyse ist mit der Rechtschreibkorrektur eines Textverarbeitungsprogramms vergleichbar.

- Syntaktische Analyse
  - überprüft, ob Quellcode ein korrektes Programm der zu übersetzenden Quellsprache ist
  - Eine lexikalisch korrekte Programm-Zeile wie z.B. `flaeche = pi * r * ;` wird als syntaktisch falsch erkannt.
  - Grammatikprüfung

- Semantische Analyse
  - überprüft über die syntaktische Analyse hinausgehende Bedingungen (z.B. Datentypen)
  - Die Anweisung `a = 3 * wurzel(5);` ist lexikalisch und syntaktisch völlig korrekt, kann semantisch dagegen falsch sein (z.B. wenn `wurzel(5)` einen String zurückgibt).

Compiler

(2) Synthesephase

- (Zwischencodegenerierung)
  - maschinennaher Code
- (Programmoptimierung)
  - Optimierung des Zwischencodes
- Codegenerierung
  - Erzeugung des ausführbaren Programms

Entwicklung von C

- integrierte Entwicklungsumgebungen (IDEs)
  - Editor
  - Compiler
  - Linker
  - Debugger

- Es empfiehlt sich, die Dateiendung .c zu verwenden, auch wenn dies bei den meisten Compilern nicht zwingend vorausgesetzt wird.
  - Makefile

GNU C Compiler

GNU C Compiler

- Teil der GCC (GNU Compiler Collection)
- populärster Open-Source-C-Compiler
- für viele verschiedene Plattformen verfügbar
- Standard-Compiler für GNU/Linux und die BSD-Varianten
- Compileraufruf: `gcc Quellcode.c -o Programm`
  (s. a. http://de.wikibooks.org/wiki/C-Programmierung → Kompilierung)
  - kompiliert und linkt "Quellcode.c„
  - Ausgabe als "Programm„
  - Flag `-c` → kein Linken
  - Informationen über weitere Parameter → *man gcc*

Portabilität von C

- **Portabilität = Übertragbarkeit**
  - Sprache selbst
  - Bibliotheken
- **Portabilität von C selbst**
  - Standards
    - *ANSI X3.159-1989 (1989)*
    - *ISO/IEC 9899:1999 –C99 (1999)*
  - Standardsprachdefinitionen
  - Standardbibliotheken / -schnittstellen
- **In der Vorlesung**
  - ANSI-C

Portabilität von C

- **Sicherung der Portabilität**
  - Keine Bibliotheken, die *von dem (Betriebs-)System* zur Verfügung gestellt werden, verwenden; ggf. eigene Bibliotheken erstellen und einbinden
  - Keine Erweiterungen des C-Sprachkerns verwenden

- **„Eingebaute“ Portabilität**
  - Aufspaltung in Sprachkern und
  - Programmbibliotheken (engl.: libraries)

- **Standardbibliothek mit rudimentären Funktionen**
  - Ein-/Ausgabe (stdio.h)
  - Dateihandling (stdio.h)
  - Zeichenkettenverarbeitung (string.h)
  - Mathematik (math.h)
  - Speicherreservierung (stdlib.h; stdio.h)
  - und einiges mehr.

Entwicklungsumgebung für C

- **Software-Installation**
  - gcc
  - make
  - git
- **Linux-Shell**
  - Empfehlung: Eine Directory pro Aufgabe

Aufgabe 02.01

Aufgabe 02.01

- **Aufgabe**
  - Gib die Zeichenkette „Hello, World!“ auf der Standardausgabe (Konsole) aus.
  - Erzeuge eine Datei hello-world.c
    - vi hello-world.c
  - Übersetze das Programm
    - make hello-world
  - Führe das Programm aus:
    - hello-world
  - Code siehe nächste Folie

Aufgabe 02.01

```c
// eineinzeiliger Kommentar

/*
    ein mehrzeiliger
    Kommentar
*/
#include<stdio.h>

int main() {

    // Ausgabe auf der Systemausgabe (\n => Zeilenumbruch)
    printf("Hallo Welt!\n");

    return 0;
}
```

Variablen, Datentypen, Operatoren und Ausdrücke

- **Variablen/Konstanten**
  - Variablen und Konstanten sind die grundsätzlichen Datenobjekte in einem Programm
- **Datentypen**
  - Vereinbarungen legen den *Typ* und den (Anfangs-)wert der Variablen/Konstanten fest
- **Operatoren**
  - kontrollieren, was mit den Werten geschieht
- **Ausdrücke**
  - Variablen und Konstanten werden mit Operatoren verknüpft
    - So werden neue Werte produziert

Variablen

- speichern Werte (im sogenannten RAM → Random Access Memory)
- können geschrieben und gelesen werden (Konstanten nur gelesen)
- Variablennamen bestehen aus Buchstaben, Ziffern und dem Unterstrich
  - Variablennamen sollten mit Kleinbuchstaben geschrieben werden
  - Konstantennamen in Großbuchstaben
  - es dürfen keine Schlüsselwörter verwendet werden (z.B. if, else, int …)

```c
#include<stdio.h>

int main() {

    int durchmesser;
    durchmesser = 5;

    printf("Der Umfang des Kreises mit Durchmesser %i beträgt:", durchmesser);

    float umfang = durchmesser * 3.1416;

    printf(" %4.2f cm", umfang);

    return 0;
}
```

Datentypen und Speicherbedarf

- C kennt nur wenige elementare Datentypen:
  - **char**, ein Byte, kann ein Zeichen aus dem Zeichensatz der Maschine aufnehmen
  - **int**, ganzzahliger Wert
  - **float**, einfach genauer Gleitkomma Wert
  - **double**, doppelt genauer Gleitkomma Wert

Zeichen

- ***char***
  - speichert Zeichen aus ASCII-Tabelle
  - jedes Zeichen besitzt einen Code (siehe nächste Folie)
  - Größe: ein Byte (Werte von -128 bis 127) oder
  - wenn nur positive Zahlen verwendet werden
    - Verwendung des zusätzlichen Schlüsselwortes ***unsigned*** (→ ***unsigned char***)
    - Wertebereich: 0 bis 255

```
#include<stdio.h>

int main() {
    char zeichen;
    zeichen = 'A';
    printf("Zeichen: %c\n", zeichen);
    zeichen = 66;
    printf("Zeichen: %c\n", zeichen);
    return 0;
}
```

```
Zeichen: A
Zeichen: B
```

Datentypen

ASCII Tabelle

Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen
	00	0	NUL		20	32	SP		40	64	@	0D	60	96	`
	01	1	SOH ^A	02	21	33	!	1E	41	65	A	1E	61	97	a
	02	2	STX ^B	03	22	34	"	30	42	66	B	30	62	98	b
	03	3	ETX ^C	29	23	35	#	2E	43	67	C	2E	63	99	c
	04	4	EOT ^D	05	24	36	$	20	44	68	D	20	64	100	d
	05	5	ENQ ^E	06	25	37	%	12	45	69	E	12	65	101	e
	06	6	ACK ^F	07	26	38	&	21	46	70	F	21	66	102	f
	07	7	BEL ^G	0D	27	39	'	22	47	71	G	22	67	103	g
0E	08	8	BS ^H	09	28	40	(	23	48	72	H	23	68	104	h
0F	09	9	TAB ^I	0A	29	41	)	17	49	73	I	17	69	105	i
	0A	10	LF ^J	1B	2A	42	*	24	4A	74	J	24	6A	106	j
	0B	11	VT ^K	1B	2B	43	+	25	4B	75	K	25	6B	107	k
	0C	12	FF ^L	33	2C	44	,	26	4C	76	L	26	6C	108	l
1C	0D	13	CR ^M	35	2D	45	-	32	4D	77	M	32	6D	109	m
	0E	14	SO ^N	34	2E	46	.	31	4E	78	N	31	6E	110	n
	0F	15	SI ^O	08	2F	47	/	18	4F	79	O	18	6F	111	o
	10	16	DLE ^P	0B	30	48	0	19	50	80	P	19	70	112	p
	11	17	DC1 ^Q	02	31	49	1	10	51	81	Q	10	71	113	q
	12	18	DC2 ^R	03	32	50	2	13	52	82	R	13	72	114	r
	13	19	DC3 ^S	04	33	51	3	1F	53	83	S	1F	73	115	s
	14	20	DC4 ^T	05	34	52	4	14	54	84	T	14	74	116	t
	15	21	NAK ^U	06	35	53	5	16	55	85	U	16	75	117	u
	16	22	SYN ^V	07	36	54	6	2F	56	86	V	2F	76	118	v
	17	23	ETB ^W	08	37	55	7	11	57	87	W	11	77	119	w
	18	24	CAN ^X	09	38	56	8	2D	58	88	X	2D	78	120	x
	19	25	EM ^Y	0A	39	57	9	2C	59	89	Y	2C	79	121	y
	1A	26	SUB ^Z	34	3A	58	:	15	5A	90	Z	15	7A	122	z
01	1B	27	Esc	33	3B	59	;		5B	91	[		7B	123	{
	1C	28	FS	2B	3C	60	<		5C	92	\		7C	124	|
	1D	29	GS	0B	3D	61	=		5D	93	]		7D	125	}
	1E	30	RS	2B	3E	62	>	29	5E	94	^		7E	126	~
	1F	31	US	0C	3F	63	?	35	5F	95	_	53	7F	127	DEL

Ganze Zahlen

- ***short (int)***
  - für kleine Zahlen
    - 16 Bit
    - Wertebereich: -32.768 bis +32.767
  - wenn nur positive Zahlen verwendet werden
    - Verwendung des zusätzlichen Schlüsselwortes ***unsigned*** (→ ***unsigned short***)
    - Wertebereich: 0 bis 65.535
  - Was passiert, wenn der Wertebereich über- bzw. unterschritten wird?
    - Dann beginnt er wieder am anderen Ende (siehe Beispiel nächste Folie)

Ganze Zahlen

- ***short (int)***

```
#include<stdio.h>
int main() {
    // kleine Zahl mit Vorzeichen deklarieren (erstellen)
    short kleineZahl;

    // kleine Zahl auf Grenzwert setzen
    kleineZahl = 32767;

    // eins hochzählen, also 32767+1, was eigentlich 32768 ergibt
    kleineZahl++;

    // kleineZahl ausgeben
    printf("Wert von kleineZahl: %i\n", kleineZahl);

    return 0;
}
```

```
Wert von kleineZahl: -32768
```

Ganze Zahlen

- ***int***
  - für gewöhnliche Zahlen
  - 16 bzw. 32 Bit
  - Wertebereich: -2.147.483.648 bis +2.147.483.647
  - mit ***unsigned*** von 0 bis 4.294.967.295

- ***long (int)***
  - für sehr große Zahlen
  - 32 Bit, bei ***long long*** 64 Bit
  - Wertebereich: -9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807
  - mit ***unsigned*** von 0 bis 18.446.744.073.709.551.615

Ganze Zahlen

```
// Dieses Programm gibt die Größen der Datentypen in Bit aus

#include<stdio.h>

int main() {
    printf("short: %2d Bit\n", sizeof(short)*8);
    printf("int: %2d Bit\n", sizeof(int)*8);
    printf("long: %2d Bit\n", sizeof(long)*8);
    printf("long long: %2d Bit\n", sizeof(long long)*8);

    return 0;
}
```

Datentypen

Kommazahlen

- ***float***
  - 32 Bit
  - Genauigkeit: 7-8 Stellen
- ***double***
  - 64 Bit
  - Genauigkeit: 15-16 Stellen
- ***long double***
  - 80 Bit – 128 Bit
  - Genauigkeit: 19-20 Stellen

```
#include<stdio.h>

int main() {
    // Deklaration von normalgroßer Kommazahl
    float kommazahl;

    // Wert zuweisen
    kommazahl = 12.3456;

    // Ausgabe
    printf("Kommazahl: %f", kommazahl);

    return 0;
}
```

```
Kommazahl: 12.3456
```

- Verwendung auch mit ***unsigned*** möglich
- Variationen je nach Plattform möglich

Deklaration

- eine Variable benennen und dem Compiler bekannt machen

```
// Variablen erstellen
int a;
int b;
int c;
```

```
Oder so (ist das Gleiche):
int a, b, c;
```

- mit dem Variablennamen wird ein bestimmter Speicherbereich angesprochen
- Variable hat zunächst einen beliebigen Wert (je nachdem, was in dem Speicherbereich steht)
- mittels Initialisierung kann die Variable nun auf einen Anfangswert gesetzt werden

Initialisierung

- Variablen sollten immer initialisiert werden

```
// Initialisierung
int a=1, b=2, c=a+b;
```

- Deklaration wird um eine Zuweisung ergänzt (Initialisierung)

Unveränderliche Variablen/Konstanten

- Konstanten sind Variablen, deren Wert nach der Initialisierung nicht verändert werden darf (Beispiel siehe nächste Folie)

Konstanten

- Variablendeklaration mit dem Schlüsselwort ***const***

```
// unveränderliche Variablen
const int RAEDER = 4;
const float PREISKARTOFFELN = 0.89;

// Hier bringt der Compiler einen Fehler oder eine Warnung
RAEDER = 5;
```

Symbolische Konstanten

- über ***#define*** -Konstruktion

```
#include<stdio.h>
#define PI 3.141592653589793

int main() { …
    float umfang = PI*5;
    …
}
```

Zuweisung

- Einfachster Operator **=**
  - setzt eine Variable auf einen bestimmten Wert
  - links vom = muss also eine Variable stehen
  - rechts vom = ein Ausdruck (z.B. Wert, Formel, Variable)

```
int a, b;

// Zuweisung eines konstanten Wertes, a ist 1
a = 1;

// Zuweisung eines Variablenwertes, b ist 1
b = a;
```

Inkrement & Dekrement

- Um den Wert einer Variablen um 1 zu erhöhen oder erniedrigen kommen die Inkrement- und Dekrement-Operatoren zum Einsatz

```
int a=0, b=0;

// a mit Inkrement-Operator hochzählen
a++;

// b ohne Inkrement-Operator hochzählen
b = b + 1;

// a dekrementieren, a ist wieder 0
a--;

// b ohne Inkrement-Operator dekrementieren
b = b - 1;
```

```
int a=0, b=0;
// Erst Zuweisung, dann Inkrement, a ist 0, b ist 1
a = b++;
a=0; b=0;
// Erst Inkrement, dann Zuweisung, a ist 1, b ist 1
a = ++b;
```

Arithmetische Operatoren

- Addition: **+**
- Subtraktion: **-**
- Multiplikation: ***
- Division: ***/***
  - bei Division von int-Variablen wird u.U. abgeschnitten
    - z.B. aus 2.1, 2.5 und 2.9 wird einfach 2
- Modulo (Rest): **%**

```
int a=0, b=2, c=5;

a = b + c; // a ist 7
a = b - c; // a ist -3
a = c / b; // a ist 2
a = c * b; // a ist 10

// Rest aus Division berechnen
a = c % b; // 5 / 2 ist 2 Rest 1, a ist 1
a = c % 3; // 5 / 3 ist 1 Rest 2, a ist 2
```

Arithmetische Operatoren

- Prioritäten

```
// Prioritäten mit Klammern setzen
a = 1 + b * c;    // Punkt vor Strich, a ist 11
a = (1 + b) * c; // 1+2 ist 3, 3*5 ist 15, a ist 15
```

- Kurzschreibweise

```
int a=1, b=2;

a += 1; // wie a=a+1 oder a++, a ist 2
a += b * 4; // wie a=a+b*4, a ist 10
a /= 2; // wie a=a/2, a ist 5
a %= 2; // wie a = a%2, a ist 1
```

Bit-Manipulation

- ***Bit-Verschiebung***
  - die Bits eines Wertes können nach links oder rechts verschoben werden
  - die Bits am Rand fallen somit raus, die Stellen auf der anderen Seite werden mit Nullen aufgefüllt
  - die Linksverschiebung geht mit <<, die Rechtsverschiebung mit >>

```
// Hier wird der Wert 5 jeweils nach links und rechts um eine Stelle verschoben.

int b=5, c, d;
c = b << 1;
d = b >> 1;
printf("c: %d, d: %d\n", c, d);
```

```
c: 10, d: 2
```

```
Zugehörige Rechnung der Bitverschiebung

b: 5 dez => 00000101 binär

c: 00000101 << 1 => 00001010 binär => 10 dezimal
d: 00000101 >> 1 => 00000010 binär => 2 dezimal
```

Bit-Manipulation

❑ *Bit-Verschiebung*

```
// Hier wird der Wert 8 um 1 nach links verschoben
int x=8, y;
y = x << 1;
printf("y= %d\n", y);
```

```
y=16
```

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	128	64	32	16	8	4	2	1
8	0	0	0	0	1	0	0	0
<<1	0	0	0	1	0	0	0	0

Typumwandlung

- ***TypeCast, Cast, Casting***
  - explizite Umwandlung von einem Datentyp in einen anderen
    - char wird intern z.B. als Zahl (ASCII) dargestellt

```
int i;

char c = 'A';
i = (int)c;
printf("char c nach int i: %d\n", i);

i = 67;
c = (char)i;
printf("int i nach char c: %c\n", c);

float f = 2.345;
i = (int)f;
printf("float f nach int i: %d\n", i);
```

```
int i = 1, j = 2;

float f = i / j;
printf("f = %f\n", f);
```

```
int i = 1, j = 2;

float f = (float) i / (float) j;
printf("f = %f\n", f);
```

```
char c nach int i: 65
int i nach char c: C
float f nach int i: 2
```

Aufgabe 02.02

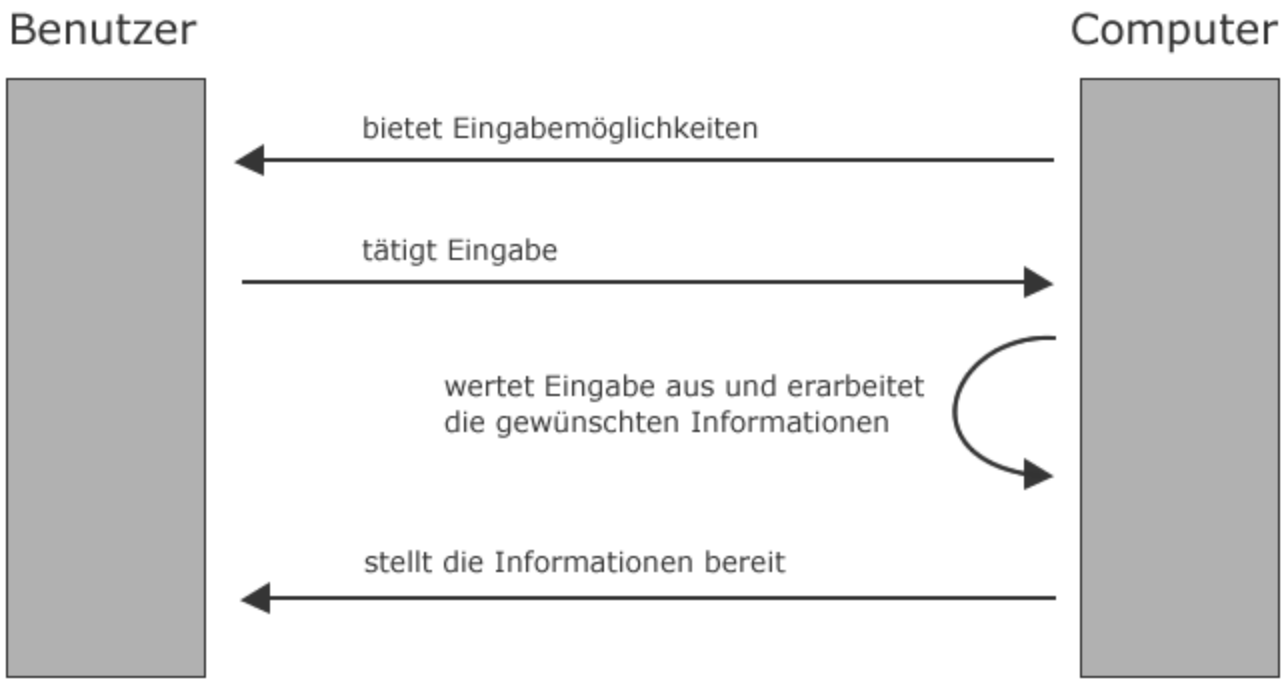
Berechne (ohne Hilfe des Computers!) die Variablenwerte!

```
int a, b=5, c, d;

a = b / 2;          // a=

c = b % 2;          // c=

d = b << a;         // d=

b = 1 - --b;        // b=


b *= -3;            // b=

d %= 3;             // d=

c += b * d + 4;     // c=

a = --b + d++;

// a=     b=      d=
```

```
a=0; b=2; c=3; d=4;

a = (b + 2) * 2 * c + 1;

// a=



a = ++b * d++ * ++c * (-1);

// a=     b=      c=      d=
```

Benutzerinteraktion

- Bildschirmausgaben haben wir bereits kennengelernt (printf)
- für eine Interaktion zwischen Benutzer und Programm sind auch Benutzereingaben notwendig



printf

http://wpollock.com/CPlus/PrintfRef.htm

- zur formatierten Ausgabe auf der Konsole

```
printf(control, arg1, arg2, …)
```

Zeichenkette mit Platzhaltern (Formatelemente) für die Argumente

Argumente 0-n

- jedes Formatelement beginnt mit dem Zeichen **%**
  - **%d** od. **%i** das Argument ist ein ganzzahliger Wert
  - **%s** das Argument ist eine Zeichenkette
    - `%-20s |hello, world |`
    - `%20s | hello, world|`
  - **%f** das Argument muss float oder double sein
    - `printf("%4.0f %6.1f", fahrenheit, celsius)`

Gleitkommazahl mit mind. 4 Zeichen breite und keine Ziffer nach dem Dezimalpunkt

Gleitkommazahl mit mind. 6 Zeichen breite und einer Ziffer nach dem Dezimalpunkt

Tastatureingaben

Wie kann ein Benutzer Daten in das Programm eingeben?

- einzelnes Zeichen einlesen
  - Mit dem Befehl getchar wird das Programm angehalten und läuft erst weiter, wenn der Benutzer eine Taste drückt
  - das eingelesene Zeichen wird in einer char Variable gespeichert

```
char c;
printf("Mit welchem Buchstaben beginnt ihr Vorname? ");
c = getchar();
printf("\nIch weiss jetzt, dass Ihr Vorname mit '%c' beginnt.\n", c);
```

```
Mit welchem Buchstaben beginnt ihr Vorname? O

Ich weiss jetzt, dass Ihr Vorname mit 'O' beginnt.
```

Tastatureingaben

- Zahlen und mehrere Zeichen einlesen
  - Verwendung von ***scanf*** (analog zur Ausgabe mit ***printf***)
  - Variablennamen in der scanf-Funktion erhalten ein kaufmännisches UND (&) vorangestellt. Warum erfahren wir auf der nächsten Folie.

```
int alter;
printf("Wie alt sind sie? ");
scanf("%d", &alter);
printf("\nIn %d Jahren sind Sie 100!\n", 100-alter);
```

```
Wie alt sind sie? 36

In 64 Jahren sind Sie 100!
```

Adressoperator

- Adressoperator „&“
  - Eine Variable kann in vier folgende Einzelteile zerlegt werden:
    1. Datentyp
    2. Name der Variable
    3. Speicheradresse der Variable
    4. Wert der Variable

```
int i;
Printf("Bitte geben Sie eine Zahl ein: ");
scanf("%d", &i);
printf("\Die Zahl, die Sie eingegeben haben war %d\n", i);
```

Datentyp	Name	Speicher-adresse	Wert
int	i	0000:123A	5

Später mehr dazu im Kapitel Zeiger/Pointer

Adressoperator

- Adressoperator „&“
  - Was beim Einlesen einer Zeichenkette richtig ist, ist bei anderen Datentypen wie Ganz- oder Gleitpunktzahlen wieder falsch:

```
/* FALSCH, da Adressoperator & fehlt */
scanf("%d", zahl);

/*
   Richtig, denn eine Zeichenkette benötigt keinen Adressoperator.
*/
scanf("%s", string);
```

Tastatureingaben

- Einlesen mit Eingabeformatierung
  - sinnvoll, wenn ein bestimmtes Eingabeformat erzwungen werden soll
    - z.B. beim Datum mit Tag.Monat.Jahr (also drei Zahlen getrennt durch einen Punkt)
    - Gültige Datumseingabe wäre z.B. 7.10.1974
    - Ungültig Eingabe: 7-10-1974

```
int tag, monat, jahr;
printf("Bitte geben Sie ihr Geburtsdatum ein [TT.MM.JJJJ]: ");
scanf("%d.%d.%d", &tag, &monat, &jahr);
printf("\nIhr internationales Geburtsdatum: %04d-%02d-%02d\n", jahr, monat, tag);
```

```
Bitte geben Sie ihr Geburtsdatum ein [TT.MM.JJJJ]: 7.10.1974

Ihr internationales Geburtsdatum: 1974-10-07
```

- Andere Datentypen einlesen
  - Analog zur Ausgabe werden beim Einlesen die gleichen Formatierungstypen verwendet
    - **%c** liest ein Zeichen ein
    - **%f** liest eine Kommazahl ein

Tastatureingaben

- Tastaturpuffer
  - Alle Eingaben landen zunächst im Tastaturpuffer, bevor sie vom Programm verarbeitet werden
  - Auch die Eingabe der ENTER-Taste landet im Puffer
    - das ist in folgendem Fall unerwünscht

```
char a, b;

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c", &a);

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c", &b);

printf("Die ASCII-Codes ihrer Zeichen sind %d und %d\n", a, b);
```

```
Geben sie ein Zeichen ein: a
Geben sie ein Zeichen ein:

Die ASCII-Codes ihrer Zeichen sind 97 und 10
```

Tastatureingaben

- Tastaturpuffer
  - Lösung: Verwenden einer zweiten Variable
    - so wird der Tastaturpuffer geleert

```
char a, b, temp;

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c%c", &a, &temp);

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c%c", &b, &temp);

printf("\nDie ASCII-Codes ihrer Zeichen sind %d und %d\n", a, b);
```

```
Geben sie ein Zeichen ein: a
Geben sie ein Zeichen ein: b

Die ASCII-Codes ihrer Zeichen sind 97 und 98
```

Tastatureingaben

- Tastaturpuffer
  - Lösung: Verwenden von **`fflush()`**
    - so wird der Tastaturpuffer geleert
    - problematisch unter Linux

```
char a, b;

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c", &a);
fflush(stdin);

printf("\nGeben sie ein Zeichen ein: ");
scanf("%c", &b);

printf("\nDie ASCII-Codes ihrer Zeichen sind %d und %d\n", a, b);
```

```
Geben sie ein Zeichen ein: a
Geben sie ein Zeichen ein: b

Die ASCII-Codes ihrer Zeichen sind 97 und 98
```

Aufgabe 02.04

Dieses Programm berechnet die Anzahl der Herzschläge seit deiner Geburt. Was muss im Programm geändert werden, damit die Gesamtzahl der Herzschläge ohne Nachkommastellen ausgegeben wird?

```
#include <stdio.h>
int main(){
    float schlaege, alter;
    printf("\n\t\tHerzschlaege\n");
    printf("\nHerzschlaege pro Minute: \n");
    scanf("%f", &schlaege);
    printf("Alter in Jahren: \n");
    scanf("%f", &alter);
    printf("\nIhr Herz hat seit Ihrer Geburt ");
    printf("%f ", schlaege * 60 * 24 * 365.25 * alter);
    printf("mal geschlagen. ");

    return 0;
}
```

Aufgabe 02.03

- Taschenrechner
  - Schreibe ein Programm, das zwei Kommazahlen einlesen kann und daraus die Summe berechnet. Die Bildschirmausgabe könnte dabei so aussehen:

```
Taschenrechner

Geben Sie die 1. Zahl ein: 16
Geben Sie die 2. Zahl ein: 5.55

Die Summe ergibt 21.55
```

Aufgabe 02.03

- Erzeuge eine neue Directory für die neue Aufgabe
  - Erzeuge zunächst ein Makefile (vi Makefile)
  - Erzeuge dann den Quellcode (vi rechner.c)
  - Führe danach das Programm aus (./rechner)

```
default: rechner

rechner.o: rechner.c
    gcc -c rechner.c -o rechner.o

rechner: rechner.o
    gcc rechner.o -o rechner

clean:
    -rm -f rechner.o
    -rm -f rechner
```


https://stackoverflow.com/questions/1484817/how-do-i-make-a-simple-makefile-for-gcc-on-linux

Aufgaben 02.05 – 02.08

Aufg. 02.05

Schreibe ein Programm, das den Benzinverbrauch eines Autos in Litern je 100 Kilometer errechnet. Als Eingabe benötigt das Programm den Benzinverbrauch insgesamt in Litern und die dazu gefahrenen Kilometer. Der Verbrauch pro 100 Kilometer ergibt sich dann aus: Liter * 100/km.

Aufg. 02.06

Lies den Radius eines Kreises ein und gib den Flächeninhalt des Kreises aus (pi*r*r).

Aufg. 02.07

Nach der Eingabe von Tagen soll angezeigt werden, wie viele Stunden, Minuten und Sekunden das **jeweils** sind.

Aufg. 02.08

Nach der Eingabe von Sekunden soll angezeigt werden, wie viele Tage und (Rest-) Stunden, (Rest-) Minuten und (Rest-)Sekunden das sind.