

## Kurseinheit 1: Einführung, Variablen und Konstanten

1. Organisation
2. Einführung in C
3. Datentypen
4. Variablen
5. Konstanten
6. Kurzeinführung Konsolenausgabe
7. Beispiel Variablen und Konstanten

## Übersicht KE 1

Lehrveranstaltung Ingenieur-Informatik – 2 SWS/2 Credits: EI1, EI+1, MKA1, MK+1, EI3nat3  
Lehrveranstaltung Programmierung 2 (Teil C) – 2 SWS/2 Credits: AI2

Unterrichtsdauer für diese Kurseinheit: 90 Minuten

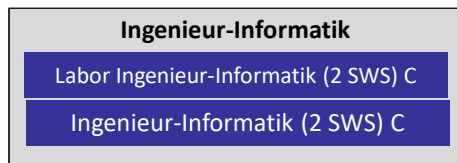
Korrespondierende Kapitel aus *C-Programmierung – Eine Einführung*: Kapitel 1 und 2

Zusatzthemen: Organisatorisches inkl. Praktikum-/Laboreinführung

## 1. Organisation

3

### Module

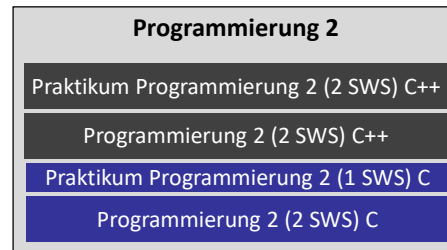


Wintersemester:  
EI1, EI+1, MKA1, MK+1, EI3nat3

Gesamtes Semester

Praktikum/Labor: ca. 4,5 Std. pro Woche für Fertigstellung der Laboraufgaben („Hausaufgaben“).

Prüfung: K90



Sommersemester:  
AI2

Erste Semesterhälfte (C)

Prüfung: K90 (C und C++)

## 1. Organisation

4

### Labor/Praktikum

Im Labor Ingenieur-Informatik / Praktikum Programmieren 2 besteht **Anwesenheitspflicht** (Liste).

Sofortiger Ausschluss aus dem Labor/Praktikum:

- Unentschuldigtes Fehlen am Labortag. Informieren Sie rechtzeitig den Dozenten vorab per Email und geben Sie das Attest im Studierendensekretariat ab. Bei (mehrfachen) Krankmeldungen entscheidet der Dozent, ob eine weitere Teilnahme noch sinnvoll ist (Wissensstand -> KR)
- Aufgaben abgeschlossener Übungseinheiten sind an folgenden Labortagen nicht vorzeigbar oder können nicht erklärt werden. Halten Sie Ihr Home-Laufwerk aktuell. Gruppenarbeiten werden nicht akzeptiert. Plagiate führen zum sofortigen Ausschluss vom Labor/Praktikum (-> o.E.)

Erfolgreiches Bestehen des Labors:

- Nachgewiesene Anwesenheit auf Teilnahmelisten
- Aufgaben aus den Übungseinheiten wurden abgenommen / nicht beanstandet
- Bestehen des Labortests am Semesterende (letzte Vorlesungswoche)

## 1. Organisation

5

### Literatur

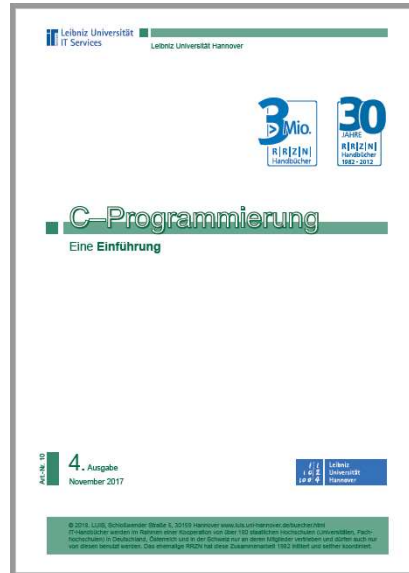
[LUH17] C-Programmierung – Eine Einführung, Leibniz-Universität Hannover, 2017.  
Download-Code kann für 4 Euro in der Bibliothek erworben werden (Bezahlung über OSKAR)

<https://www.luis.uni-hannover.de/e-books.html>  
<https://handbuch.luis.uni-hannover.de/download/>

Das Vorlesungsskript (Präsentation) (Moodlekurs Ingenieur-Informatik) orientiert sich an diesem Dokument. Ergänzend werden im Vorlesungsskript zusätzliche Themen behandelt (z.B.):

1. Statische und dynamische Bibliotheken
2. Threading
3. Komplexitätsmetriken

...

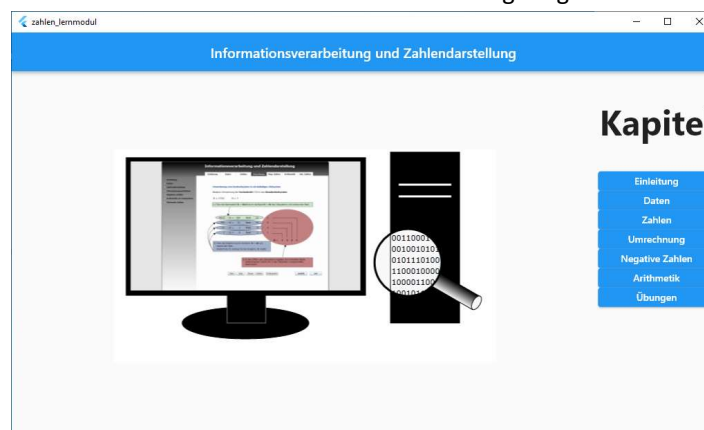


## 1. Organisation

6

### E-Learning Modul

Das wichtige Thema Informationsverarbeitung und Zahlendarstellung ist im Rahmen eines E-Learning Moduls selbst zu erarbeiten! Wird im Labortest abgefragt!



Unter Moodle verfügbar (zip-Datei, Release\zahlen\_lernmodul). Das E-Learning Modul wird am ersten Labortag gezeigt! Es kann direkt auch aus der VM gestartet werden.

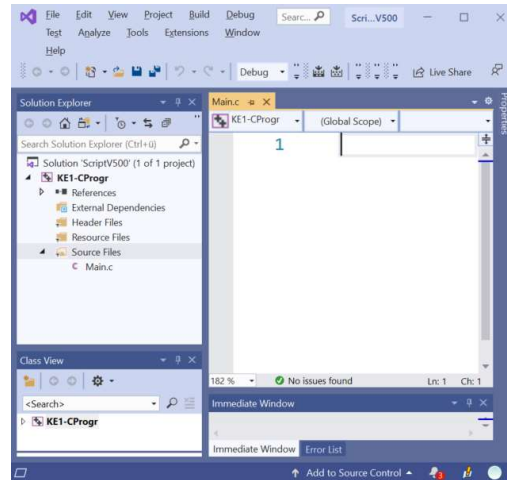
## 1. Organisation

7

### Entwicklungsumgebung

Im Rahmen dieser Lehrveranstaltung wird Microsoft Visual Studio (Community Edition) in der neuesten Version eingesetzt. Diese Entwicklungsumgebung (engl. Integrated Development Environment – IDE) ist unter BWLehrpool installiert.

Installieren Sie sich die IDE auf Ihrem eigenen Rechner. Dazu benötigen Sie einen Microsoft Account. Bei der Installation ist „Desktopentwicklung mit C++“ zu installieren. Wählen Sie als Sprachpaket „Englisch“ aus. Sollten Sie im Labor/Praktikum mit Ihrem eigenen Laptop arbeiten, so kann nur Unterstützung beim Sprachpaket „Englisch“ gegeben werden.



## 1. Organisation

8

### C-Coding Styleguide

C-Coding Styleguide

Ingenieur-Informatik (II), Programmieren 2, Embedded Systems 1+2 und Embedded Echtzeitsysteme (E2S)

1

#### C-Coding Styleguide

-Programmierreichtlinien-

Hochschule Offenburg  
Fakultät EMI

Lehrgebiete:  
Ingenieur-Informatik  
Embedded Systems 1+2  
Embedded Echtzeitsysteme

Prof. Dr.-Ing. Daniel Fischer  
Dipl.-Ing. (FH) Alexander Badura

Ergeben sich bezüglich den Lehrgebieten unterschiedliche Programmierreichtlinien, werden die beiden Regelvarianten der Richtlinie mit II (Ingenieur-Informatik) und ES (Embedded Systems 1+2, Embedded Echtzeitsysteme) kenntlich gemacht. Programmierreichtlinien, die nur für Embedded Systems 1+2 und Embedded Echtzeitsysteme relevant sind, sind ebenso mit ES gekennzeichnet.

Zur Benennung einzelner Metriken kommt das SW-Werkzeug Testwell CMT++ zur Anwendung. Dieses SW-Werkzeug ist in der zu verwendenden VM (BWLehrpool) installiert und dient dem Metriken, wie z.B. LOCpro (Lines Of Code - program), c% (Kommentardichte) und v(G) (Zyklomatische Komplexität - Cyclomatic Complexity CC - nach McCabe) zu bestimmen. Empfohlene Regeln (recommended) sind in der Nummernspalte grn hinterlegt.

Hochschule Offenburg, Fakultät EMI

V3 0.6, 16.05.2020

Allgemeines	
A1	Der Code-Editor ist so zu konfigurieren, dass ein Tabulator durch drei Leerzeichen ersetzt wird. Damit ist das Erscheinungsbild vom Sourcecode gleich, wenn dieser in verschiedenen Editoren geöffnet wird.
A2	Sourcecode wird in englischer Sprache verfasst. Das gilt z.B. für Kommentare, Variablen- und Funktionsnamen.

Der C-Coding Styleguide legt fest, wie der Programmcode genau geschrieben werden muss. Dieser ist im Labortest und in der Prüfung anzuwenden (darf mitgebracht werden). Neueste Version befindet sich im Moodlekurs.

## 2. Einführung in C

9

### Einsatzgebiete von C

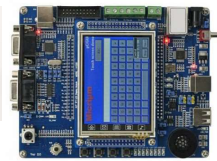
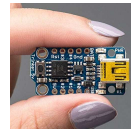


Betriebssysteme sind in C programmiert.

Oft wird ein solches System in C und C++ programmiert, wobei dann die systemnahen Komponenten in C erstellt werden.



In vielen Kleingeräten (Waschmaschine, Kaffeemaschine, ...), Steuerungen (Heizungen, Aufzüge, ...), Sensoren und Fahrzeugen sind sogenannte Embedded Systems (Microcontroller) verbaut. Diese werden meist noch in C programmiert.



## 2. Einführung in C

10

### Standardisierung

Standard	Bemerkung
K&R-C	Diese Version basiert hauptsächlich auf dem ersten Buch zu C von den beiden Autoren Kernighan und Ritchie von 1978.
C89	Die erste echte Standardisierung erfolgte über das American National Standards Institute (ANSI) im Jahre 1989.
C90	Ein Jahr nach dem Erscheinen des 1. Standards wurden kleine Änderungen hinzugefügt und die ISO-Norm C90 definiert. Sie ist Basis vieler heutiger C-Implementierungen. Die wichtigsten Verbesserungen waren die Einführung von Funktionsprototypen sowie die Normierung der C-Standardbibliothek.
C95	1995 wurden in einem neuen Standard Fehlerbehebungen, einige neue Makros sowie die Unterstützung weiterer Zeichensätze zusammengefasst. Obwohl dieser Standard schon relativ alt ist, wird er selten von den gängigen Compilern vollständig implementiert.
C99	Über die Jahre wurden einige häufig vermisste Sprachkonstrukte und Schreibweisen anderer Sprachen hinzugefügt wie der Datentyp <code>_Bool</code> , der einzeilige C++-Zeilenkommentator <code>/// //</code> oder die Möglichkeit, Variablen direkt in einer <code>for</code> -Schleife zu definieren.
C11	Ende 2011 wurde der aktuelle C11-Standard verabschiedet. Er enthält Korrekturen der Vorversion und Neuerungen wie beispielsweise Unterstützung von Multithreading, neue Datentypen und generische Ausdrücke.

[LUH17]

## 2. Einführung in C

11

### Algorithmus

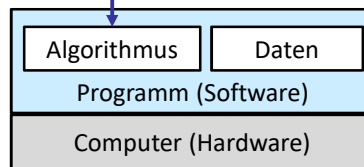
„Fährt am Freitag  
ein Bus von Offenburg  
nach Stuttgart?“



Konkrete  
Problemstellung  
Reale Welt



Der notwendige **Algorithmus** ist in einer  
Programmiersprache (z.B.) zu realisieren.



Abstraktion der Problem-  
stellung und Lösung



Ja!  
20.12 Uhr am  
Hbf. OG  
Ticket: 25 Euro

Lösung

Ein Algorithmus ist somit eine Folge von Anweisungen, die unter gleichen Ausgangswerten nach endlich vielen Schritten in einer endlichen Zeit das gesuchte Ergebnis ermitteln.  
[LUH17]

## 2. Einführung in C

12

### Schlüsselwörter

Jede Programmiersprache verfügt über feste Menge von Schlüsselwörtern. In C gibt es die folgenden Schlüsselwörter (**C89**):

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	

Schlüsselwörter ab **C99**:

_Bool	_Complex	_Imaginary	inline	restrict
-------	----------	------------	--------	----------

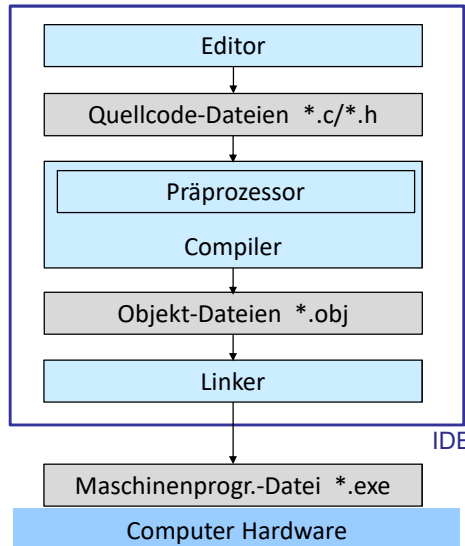
Schlüsselwörter ab **C11**:

_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			

## 2. Einführung in C

13

### Erzeugung von ausführbaren Programmen



Ein **Editor** ist ein Programm, mit dem Textdateien aber auch Quellcode-Dateien erstellt werden können (z.B. \*.txt).

**Präprozessor** ist C-spezifisch – Anweisungen beginnen mit # (siehe später).

In C kommt ein Compiler zur Anwendung.

**Compiler** übersetzt das Programm vollständig. Bei einem Programmfehler wird keine Maschinenprogramm-Datei (\*.exe) generiert.

**Interpreter** übersetzt das Programm immer zeilenweise und führt diese Zeile aus. Bei einem Programmfehler wird die Ausführung bei der fehlerhaften Zeile abgebrochen.

**Linker** bindet alle Objektdateien zusammen. Details später in Embedded Systems 1.

## 2. Einführung in C

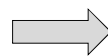
14

### Das erste C-Programm

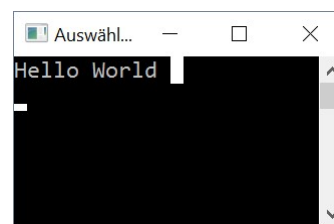
```
#include <stdio.h>

int main(void)
{
    //Prints "Hello World" to Console
    printf("Hello World\n");

    //Returns 0 to OS signaling
    //that all is okay
    return 0;
}
```



Konsole



Details siehe 1.5 aus [LUH17]

Der in Visual Studio integrierte Editor verwendet unterschiedliche Farben für Schlüsselwörter (Blau), Kommentar (Grün), Zeichenketten (Rot), Präprozessor-Anweisungen (Grau), Bibliotheken (Rot), Funktionsnamen/-aufrufe (Schwarz) usw.

Im Rahmen dieser Lehrveranstaltung werden nur **Konsolenprogramme** geschrieben. Fensterbasierte Anwendungen (Graphical User Interfaces – GUI) werden hier nicht behandelt.

### 3. Datentypen

15

#### Was ist ein Datentyp?

Zusammenfassung konkreter **Wertebereiche** und darauf **definierter Operationen** zu einer Einheit. Beispiele können Ganz- oder Kommazahlen, Zeichenketten oder auch komplexere Typen wie Datum/Zeit oder Objekte sein.

Quelle: *de.wikipedia.org*

Wie in der Mathematik, so gibt es auch beim Programmieren Variablen und Konstanten. Ein Beispiel aus der Mathematik ist die folgende Gleichung:

$$y = x + 2 + C$$

↑    ↑    ↑    ↑

Variablen    Konstanten (literale Konstante und konstante Variable)

Variablen und Konstanten müssen in C einem Datentyp zugewiesen werden, da C eine (schwach-) typisierte Programmiersprache ist.

Jedem Datentyp liegt ein bestimmter **Wertebereich** zugrunde. Dies hängt von der Speichergröße des Datentyps in Byte ab (siehe E-Learning Modul).

### 3. Datentypen

16

#### Überblick: Basisdatentypen

Die Programmiersprache C (**C89**) enthält die folgenden Basisdatentypen:

Zeichendatentyp:	<b>char</b>
Ganzzahldatentyp:	<b>int</b>
Fließkommatypen:	<b>float</b>
	<b>double</b>

Durch weitere Schlüsselwörter (Modifizierer) wie **short**, **long**, **signed** und **unsigned** können die Basisdatentypen in ihrem Wertebereich und teilweise auch in ihrer Genauigkeit verändert werden.

Erst ab **C99** gibt es in C einen Boolean Datentyp **\_BOOL** (true, false). In C ist daher ein Wert ungleich 0 immer true und ein Wert gleich 0 bedeutet immer false. Ebenso gibt es ab C99 Datentypen für komplexe Zahlen **\_Complex** und rein imaginäre Zahlen **\_Imaginary**.

Für diese Basistypen existieren die Operationen +, -, /, \* sowie der Zuweisungsoperator =. Achtung: Zuweisungsoperator ist nicht das „Gleichzeichen“ aus der Mathematik!



### 3. Datentypen

17

#### Kurzüberblick Bit und Byte

Ein **Bit** (Binary Digit) stellt die kleinste Informationseinheit dar. Ein Bit kann eine 0 oder eine 1 enthalten (zwei unterschiedliche Zustände).

1 Bit	Zwei Zustände	0, 1
2 Bit	Vier Zustände	00, 01, 10, 11
3 Bit	Acht Zustände	000, 001, 010, 011, 100, 101, 110, 111
...		
8 Bit	256 Zustände	00000000, ..., 11111111

Um schneller Daten verarbeiten zu können, werden intern im Rechner Bits zusammengefasst. 8 Bits werden als ein **Byte** bezeichnet.

Moderne Rechner können mehrere Bytes gleichzeitig verarbeiten (64-Bit Rechner = 8-Byte Rechner).

### 3. Datentypen

18

#### Zeichendatentypen char

Datentyp	Größe	Wertebereich
char	1 Byte	signed char oder unsigned char
signed char	1 Byte	-128 bis 127 ( $-2^7$ bis $2^7-1$ )
unsigned char	1 Byte	0 bis 255 (0 bis $2^8-1$ )

Modifizierer hier: **signed** (Vorzeichen) und **unsigned** (kein Vorzeichen)

Der C-Standard legt **nicht** fest, ob es sich bei einem char um ein signed char oder unsigned char handelt. Dies implementiert jeder Compilerhersteller unterschiedlich.

Die Asymmetrie des Wertebereichs bei signed char (und allen anderen signed-Datentypen) entsteht durch die interne Zweierkomplement-Darstellung von negativen Zahlen. Siehe hierzu auch das E-Learning Modul.

### 3. Datentypen

19

#### Zeichendatentyp – ASCII-Code

American Standard Code for Information Interchange

(Standard-)ASCII-Code:

- Nicht druckbare Zeichen (0-31)
- Druckbare Zeichen (32-127)

Mit 7 Bits lassen sich 128 unterschiedliche Zeichen codieren (0-127).

Erweiterter ASCII-Code:

Mit einem weiteren Bit (7 + 1) lassen sich 256 Zeichen codieren. Zeichen 128-255 sind dabei je nach Zeichensatz anders.

Für andere Zeichensätze (Arabisch, Chinesisch) reicht ein Byte nicht mehr aus. Hier werden dann „Wide Character“ Datentypen verwendet (2 Byte, 16 Bit)

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

### 3. Datentypen

20

#### Ganzzahldatentypen int (32-Bit System)

Datentyp	Größe auf 32-Bit System	Wertebereich
short int	2 Byte	-32768 bis 32767 ( $-2^{15}$ bis $2^{15}-1$ )
unsigned short int	2 Byte	0 bis 65535 (0 bis $2^{16}-1$ )
int	4 Byte	-2147383648 bis 2147383647 ( $-2^{31}$ bis $2^{31}-1$ )
unsigned int	4 Byte	0 bis 4294967295 (0 bis $2^{32}-1$ )
long int	4 Byte	-2147383648 bis 2147383647 ( $-2^{31}$ bis $2^{31}-1$ )
unsigned long int	4 Byte	0 bis 4294967295 (0 bis $2^{32}-1$ )
long long int	8 Byte	... ( $-2^{63}$ bis $2^{63}-1$ )
unsigned long long int	8 Byte	... (0 bis $2^{64}-1$ )

Modifizierer hier: **unsigned** sowie **short** und **long**

Der C-Standard legt fest, dass es sich um einen signed Datentypen handelt, falls der Modifizierer unsigned nicht angegeben wird (Default-Verhalten).

### 3. Datentypen

21

#### Gleitkommatypen

Datentyp	Größe	Wertebereich
float	4 Byte	$\pm 1,18 \cdot 10^{-38}$ bis $\pm 3,4 \cdot 10^{38}$
double	8 Byte	$\pm 2,23 \cdot 10^{-308}$ bis $\pm 1,79 \cdot 10^{308}$
long double	10, 12 oder 16 Byte	Abhängig von der Größe

Die interne Darstellung einer Fließkommazahl ist standardisiert (IEEE 754). Dies wird im Rahmen der Vorlesung Embedded Systems 1 im Detail behandelt.

Die Gleitkommatypen können auch noch spezielle Werte wie  $\pm 0$ ,  $\pm \infty$  oder NaN (Not a Number) abbilden.

### 3. Datentypen

22

#### Abhängigkeit von der Plattform

Je nach (Hardware-)Plattform und Compiler unterscheiden sich die Größen einiger Datentypen. Der Zeichendatentyp (char, unsigned char, signed char) hat immer die Größe 1. Ebenso ist die Größe der Gleitkomma-Datentypen float und double definiert.

Letztendlich gibt der C Standard nur vor:

$1 == \text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short int}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long int}) \leq \text{sizeof}(\text{long long int})$ .

Bei **sizeof** handelt es sich um einen Operator, der die Größe eines Datentyps zurückliefert.

Im Rahmen der Vorlesung und in der Prüfung wird von einem 32-Bit System (genauer einem **ILP32**: int, long und Pointer sind 32 Bit) ausgegangen.

In der Headerdatei **limits.h** finden sich die jeweiligen maximalen und minimalen Werte eines Datentyps auf der verwendeten Plattform.

## Definition einer Variablen

Um Variablen zu verwenden, müssen diese in C erst definiert werden. Bei einer Definition wird auch automatisch Speicherbereich zur Verfügung gestellt. Bei einer Deklaration wird nur die Variable bekannt gemacht, aber kein Speicher zur Verfügung gestellt.

**Definition = Deklaration + Speicher wird zur Verfügung gestellt.**

Variablen sollten vor ihrer erstmaligen Anwendung auch mit einem Wert vorbelegt werden (Initialisierung).

```
unsigned int uiDayInMonth = 24U;    U siehe später
```

Datentyp      Name der Variablen      Initialisierung

Der Name einer Variablen (Bezeichner) beginnt immer mit einem Buchstaben oder Unterstrich. Klein- und Großbuchstaben werden unterschieden. Umlaute und ß sind **nicht erlaubt**. Der C-Coding Styleguide gibt genau vor, wie der Name der Variablen zu lauten hat. Im Variablennamen ist jetzt auch der Datentyp (hier ui für unsigned int) zu hinterlegen. Die Regeln finden Sie im C-Coding Styleguide (DV3 II).

## Wertzuweisung – Zuweisungsoperator =

Wird einer Variablen bei der Definition ein Wert zugewiesen, dann handelt es sich um eine Initialisierung. Im Programmablauf kann aber auch ein neuer (oder neu berechneter) Wert einer Variablen zugewiesen werden. Dies wird als Wertzuweisung verstanden. Der Operator wird Zuweisungsoperator genannt. Symbol ist das =.

```
unsigned int uiA = 24U;  
unsigned int uiB = 12U;  
unsigned int uiC;
```

### 1. Operation: Addition

```
uiC = uiA + uiB;
```

## 2. Operation: Zuweisung

Die folgende Gleichung ist mathematisch falsch, als C-Anweisung aber korrekt. Andere Programmiersprachen wie Pascal, Delphi und ST verwenden daher := als Zuweisungsoperator, um den Unterschied zwischen Wertzuweisung und Gleichheit besser zu unterscheiden.

```
i = i + 1;
```

## 4. Variablen

25

### Weitere Modifizierer

Die beiden weiteren Modifizierer

#### **volatile** und **register**

sind insbesondere im Bereich der Embedded Systems sehr wichtig und werden dort im Detail behandelt (Lehrveranstaltung Embedded Systems 1). Diese Schlüsselwörter werden vor die Datentypen geschrieben und beeinflussen die **Codeoptimierung**.

Kurze Beispiele:

```
volatile int iGlobalVar1 = 0;
```

```
register int iLocalVar1 = 42;
```

## 5. Konstanten

26

### Unterscheidung

Der Übergriff „Konstante“ ist in **literale Konstanten**, **konstante Variablen** und **symbolische Konstanten** zu unterscheiden.

**Literale Konstanten** wären z.B. 3.1415 oder 42 oder 24U oder "Hello World\n "

```
unsigned int uiDayInMonth = 24U;
```

**Konstante Variablen** sind Variablen deren Werte zur Laufzeit nicht mehr geändert werden können. Hierbei wird das Schlüsselwort **const** eingesetzt.

```
const float cfPi = 3.1415F;
```

**Symbolische Konstanten** werden durch den Präprozessor durch literale Konstanten ersetzt. Dies wird in einer späteren Kurseinheit noch ausführlich behandelt.

```
#define MAX_ELEMENTS 100
```

## 5. Konstanten

27

### Literale Konstanten

#### Integer-Konstanten

##### Dezimalsystem

73    42    -47    47U  
4294967295UL    2147383647L

##### Hexadezimalsystem

0xBADDCAFE    0xFE    0x0

##### Oktalsystem – Empfehlung: nicht verwenden

047    01234567

#### Gleitkomma-Konstanten

3.1415    3.1415F  
double    float

#### Zeichen-Konstanten

'X'    'x'    '1'    '-'    '@'

##### Druckbare Zeichen

CRLF    TAB    Bell  
'\n'    '\t'    '\b'

##### Nicht druckbare Zeichen (ASCII-Code 0-31)

#### Zeichenketten-Konstanten

"Hello"    "Dies ist \"wichtig\"!"

Zeichenketten können auch nicht druckbare Zeichen enthalten.

"\n\tHello\nWorld\n"

Ein \ muss hier als \\ angegeben werden.

"c:\\temp"

## 5. Konstanten

28

### Konstante Variablen

Konstante Variablen sind Variablen deren Werte zur Laufzeit nicht mehr geändert werden können. Hierbei wird das Schlüsselwort **const** eingesetzt.

```
const double cdPI = 3.1415;  
const unsigned int cuiMaxWheels = 8U;
```

Wird bei der Definition der konstanten Variablen die Initialisierung vergessen, so ergibt sich ein Compilerfehler.

```
const signed char cscGroup; // Compiler Error
```

Konstante Variablen sind literalen Konstanten vorzuziehen, wenn diese mehrfach im Programm vorkommen. So ist sichergestellt dass immer der gleiche Wert verwendet wird! Welcher Wert soll für Pi verwendet werden?

3.1415 oder 3.141592 oder 3.141592653589793238462381

Wichtig ist, dass im gesamten Programm der gleiche Wert verwendet wird.

## 6. Kurzeinführung Konsolenausgabe

29

### Beispiele

```
#include <stdio.h>
//...
printf("Hello World\n");
//...
```

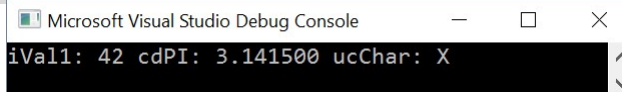
Hier die Standard Input/Output-Bibliothek eingebunden.

Jetzt kann die Funktion printf (Konsolenausgabe) verwendet werden.

Neben statischen Text will man häufig auch Werte von Variablen oder Konstanten ausgeben. Dazu muss in den Text **Formatbeschreiber** hinzugefügt werden.

```
#include <stdio.h>
//...
int iVal1 = 42;
const double cdPI = 3.1415;
unsigned char ucChar = 'X';
//...
printf("iVal1: %d cdPI: %f ucChar: %c\n", iVal1, cdPI, ucChar);
//...
```

Um ein % als Text auszugeben, muss ein %% geschrieben werden!



Microsoft Visual Studio Debug Console

iVal1: 42 cdPI: 3.141500 ucChar: X

## 6. Kurzeinführung Konsolenausgabe

30

### Wichtige Formatbezeichner für printf

Die wichtigsten Formatbezeichner für printf sind:

%i oder %d	int (vorzeichenbehaftet)
%c	char
%f	float <b>und</b> double (C89)
%e oder %E	float <b>und</b> double in Exponentialformat
%s	Zeichenkette
%x oder %X	Hexadezimalzahl (Ziffernbuchstaben groß (%X) oder klein (%x))

## 7. Beispiel Variablen und Konstanten

31

### Probleme bei unpassenden Datentypen (Variablen und Konstanten)

```
#include <stdio.h>

int main(void)
{
    unsigned char ucChar2;
    const unsigned char cucChar3 = '?';
    int iVal2;
    const unsigned int cuiVal3 = 4294967295U;

    ucChar2 = cucChar3; // Assignment const variable
    ucChar2 = 47U;      // Assignment literal const
    ucChar2 = -47;      // Warning: Mismatch
    iVal2 = 4711;       // Assignment literal const
    iVal2 = cuiVal3;    // No Warning: but still mismatch

    printf("Zeichen ucChar2: %c (ASCII-Code: %d) iVal2: %d",
           ucChar2, ucChar2, iVal2);

    return 0;
}
```

warning C4245: '=': conversion from 'int' to 'unsigned char', signed/unsigned mismatch

Microsoft Visual Studio Debug Console

Zeichen ucChar2: Ø (ASCII-Code: 209) iVal2: -1

## Zusammenfassung KE 1

32

### Behandelte Schlüsselwörter in KE 1

Schlüsselwörter C89:

auto	do	goto	signed✓	unsigned✓
break	double✓	if	sizeof✓	void
case	else	int✓	static	volatile✓
char✓	enum	long✓	struct	while
const✓	extern	register✓	switch	
continue	float✓	return	typedef	
default	for	short✓	union	

Schlüsselwörter ab C99:

_Bool✓	_Complex✓	_Imaginary✓	inline	restrict
--------	-----------	-------------	--------	----------

Schlüsselwörter ab C11:

_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			