Karlsruher Institut für Technologie

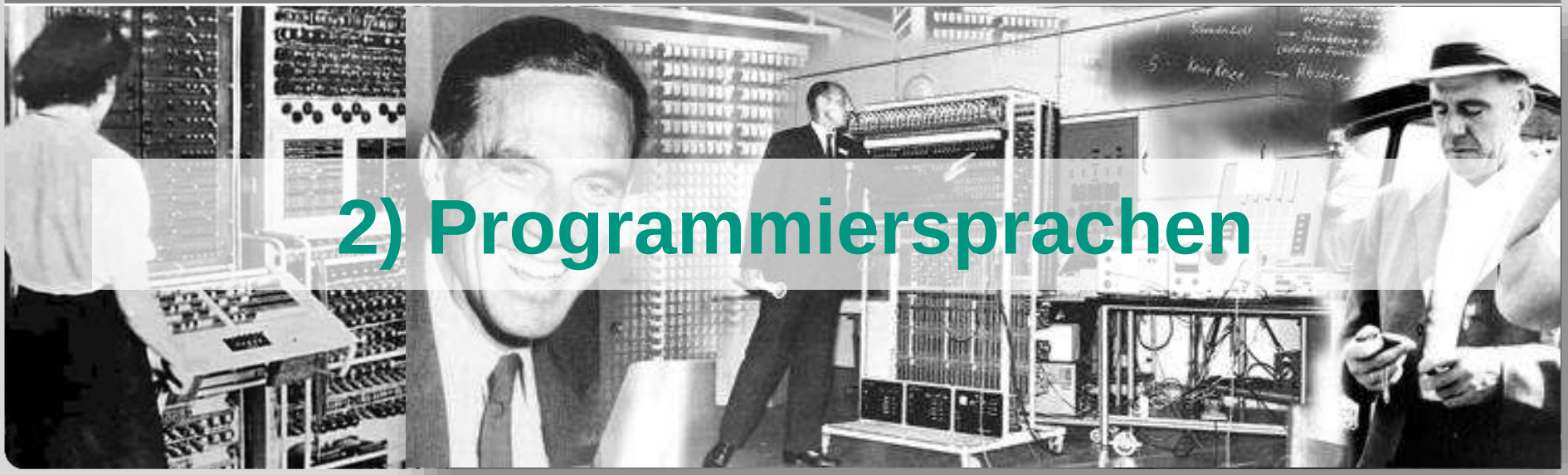
Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Vorlesung Informationstechnik (IT)

Sommersemester 2018

Institut für Technik der Informationsverarbeitung (ITIV)



2) Programmiersprachen

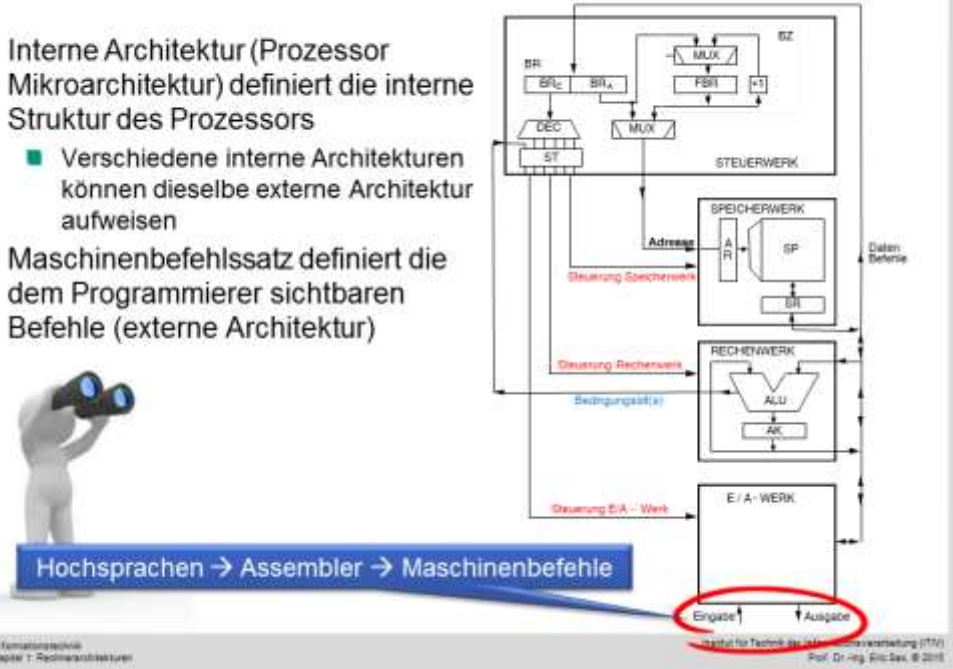
IT Inhalt

1. **Einleitung und Motivation** ✓
   - Organisatorisches
   - Begriffe
   - Typische Anwendungen
2. **Programmiersprachen**
   - Klassifikationen und Software-Komponenten (aus DT)
   - Was ist eine Programmiersprache?
   - Formale Sprachen
   - Höhere Sprachen
   - Programmierparadigmen
   - Wichtige Programmiersprachen

Maschinenbefehlssatz

Sicht des Programmierers

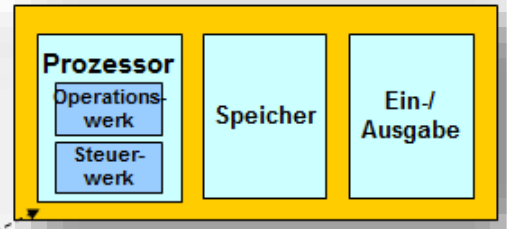
- Interne Architektur (Prozessor Mikroarchitektur) definiert die interne Struktur des Prozessors
  - Verschiedene interne Architekturen können dieselbe externe Architektur aufweisen
- Maschinenbefehlssatz definiert die dem Programmierer sichtbaren Befehle (externe Architektur)



Hochsprachen → Assembler → Maschinenbefehle

Rechnerarchitekturen

- Interne Architektur



„Rechnerkern“,

Maschinenbefehlssatz

- Maschinenbefehlssatz definiert die Programmierern sichtbaren Befehle
  - Instruction Set Architecture ISA (externe Architektur)
- Häufig wird ISA mit der Prozessor Architektur gleichgesetzt
- Maschinenbefehlsatz definiert die dem Programmierer sichtbaren Befehle (externe Architektur)

Programmiersprachen

- Computer besitzen einen binären Maschinen-Befehlssatz
- Programmiersprachen bieten die Möglichkeit, den Computer auf höherer Ebene zu programmieren (abstrakter, verständlicher für den Menschen, weniger Fehleranfällig)
- Geschaffen, um Entwicklung von Software effizienter zu machen

C Code

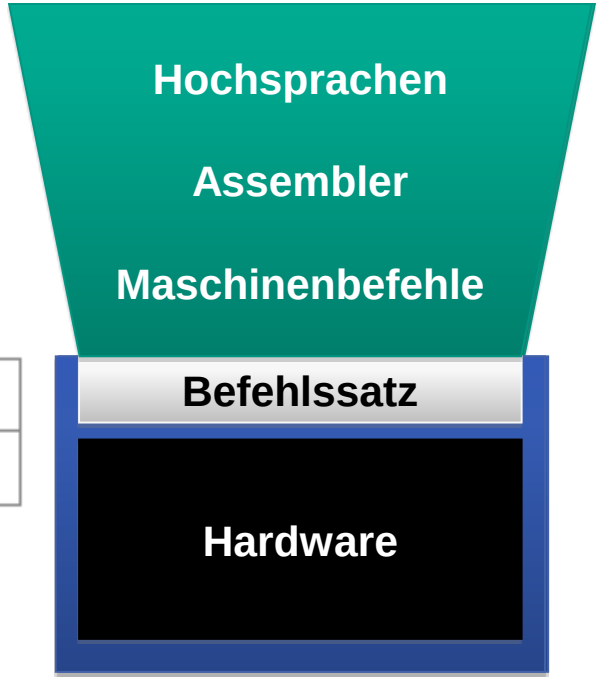
```
void main( void )
```

Assembly Language

```
MOV  A,#02
```

Machine Language Instruction

0	1	0	1	0	0	0	1
0	0	0	0	0	0	1	0



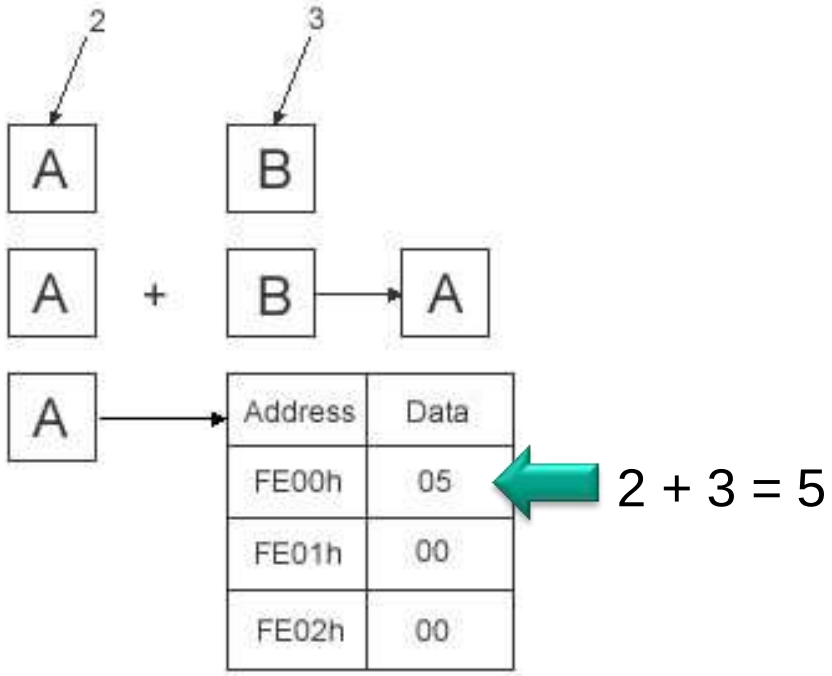
Software

externe Architektur

interne Architektur

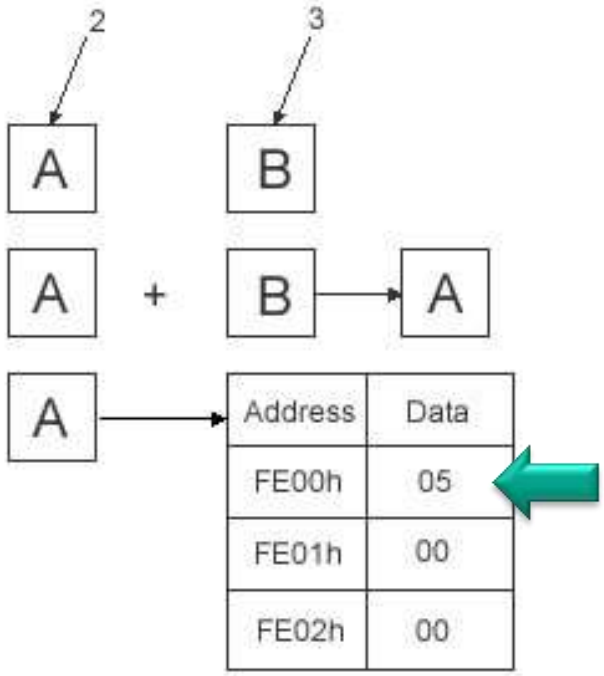
Assembly Code vs. C-Code

Assembly Code	Meaning
MOV A,#02	Write the numerical value 2 into Register A.
MOV B,#03	Write the numerical value 3 into Register B.
ADD A,B	Add the values in Registers A and B, and write the sum into Register A.
MOV !FE00h,A	Write the content of Register A into Address FE00



Assembly Code vs. C-Code

Assembly Code	Meaning
MOV A,#02	Write the numerical value 2 into Register A.
MOV B,#03	Write the numerical value 3 into Register B.
ADD A,B	Add the values in Registers A and B, and write the sum into Register A.
MOV !FE00h,A	Write the content of Register A into Address FE00



C Code

```
void main( void ) {
        int    i ;
        i = 2+3;
}
```

Was ist eine Programmiersprache?

- Programmiersprache: Notation für ein Computerprogramm
  - Darstellung während und nach der Entwicklung (Programmierung)
  - Übermittlung des resultierenden Programms zur Ausführung an Rechensystem (s. „Programmcode in Hauptspeicher“)
- Programmiersprache muss für maschinelle Analyse geeignet sein (zahlreiche Einschränkungen)
  - Programmiersprachen basieren auf „formalen Sprachen“
- Programmiersprache muss für Menschen lesbar sein
  - Im Gegensatz zu einem Format
  - Im Quelltext ist ein in einer Programmiersprache geschriebener Text und somit die lesbare und schreibbare Sicht auf ein Computerprogramm.
- *Konsequenz:*
  - *Programmiersprache kommt die Rolle einer „Lingua Franca“ zu, die sowohl der menschliche Programmierer als auch der Computer versteht.*

Fremde Sprache …

Programm

- Ein Programm ist eine Folge von elementaren Schritten.
  - Ein typischer elementarer Schritt ist die Wertzuweisung.
- Komplexere Anweisungen werden aus den elementaren Anweisungen zusammengesetzt.
- Die Reihenfolge der Schritte ergibt sich aus Kontrollstrukturen wie
  - Konkatenation, Sequenzen `(…; …; …;)`
  - Alternativen `(if - then - else, switch)`
  - Iterationen `(for, while Schleifen)`
  - Unterprogrammaufrufe `(call / x() )`
- Die Speicherung veränderlicher Daten erfolgt in Variablen.
- Es gibt elementare Datentypen
  - `Integer, Real, Boolean, Char, String, Array …`
  - Daraus können komplexere Datentypen zusammengesetzt werden

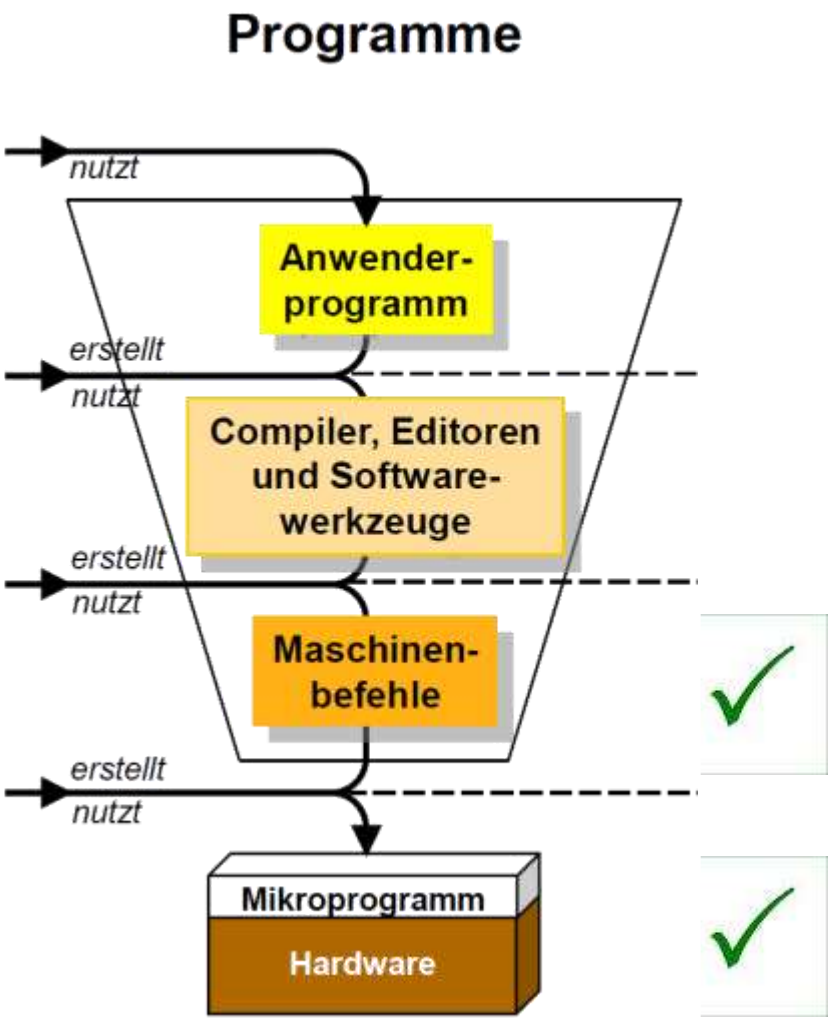
Sichtweise auf Programme

Anwender

Anwendungs-
programmierer

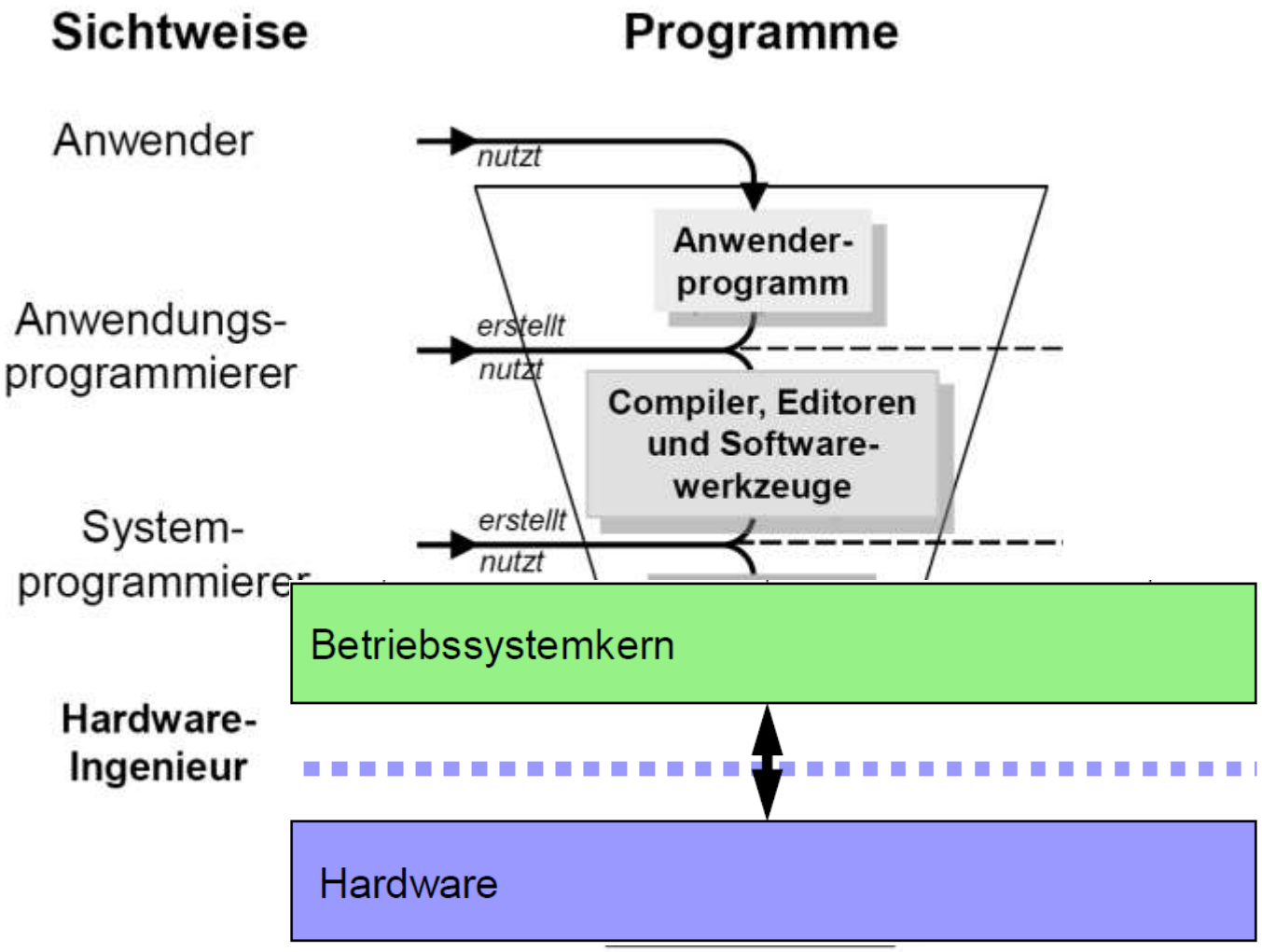
System-
programmierer

Hardware-
Ingenieur



Quelle: KIT IMI

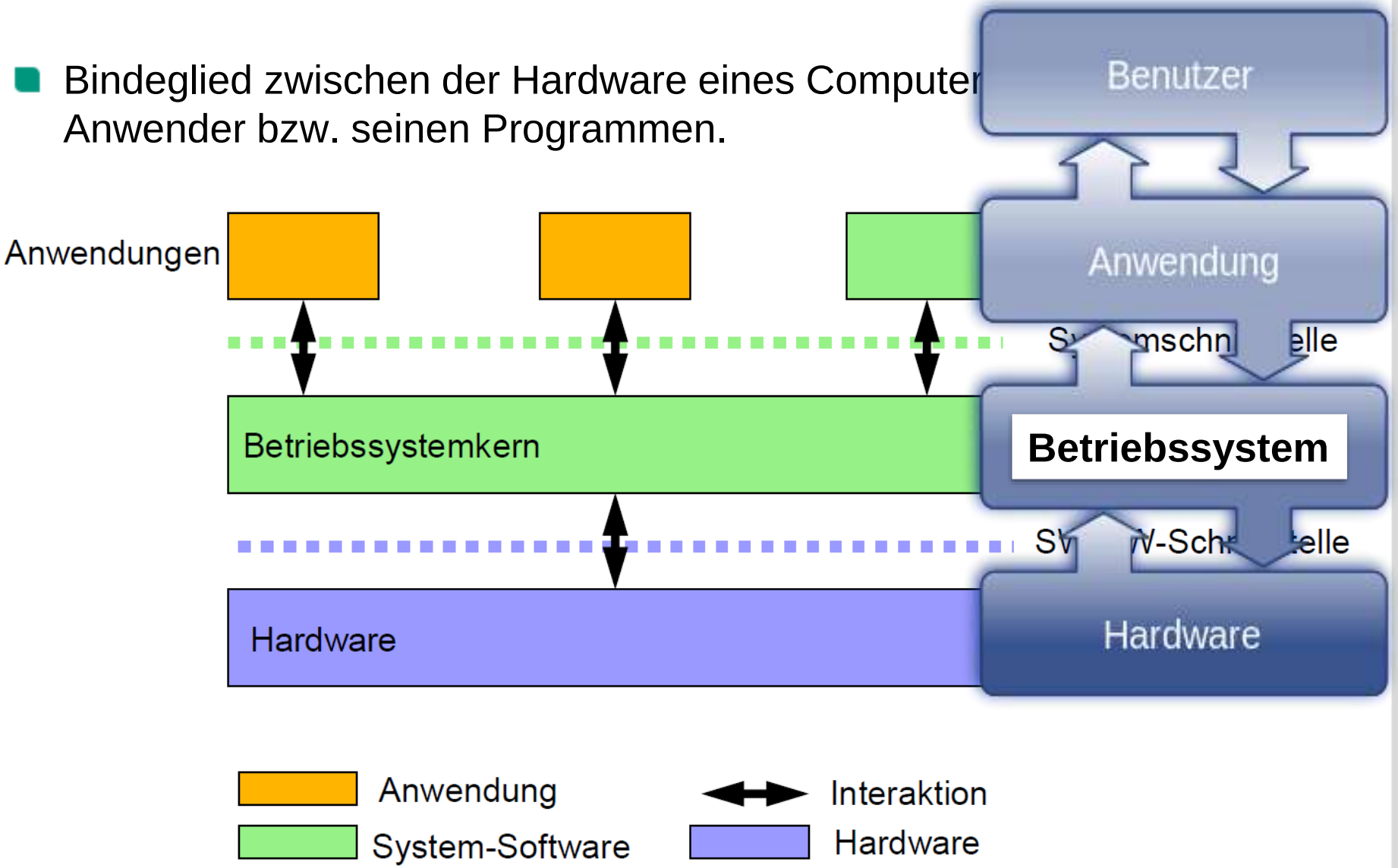
Sichtweise auf Programme



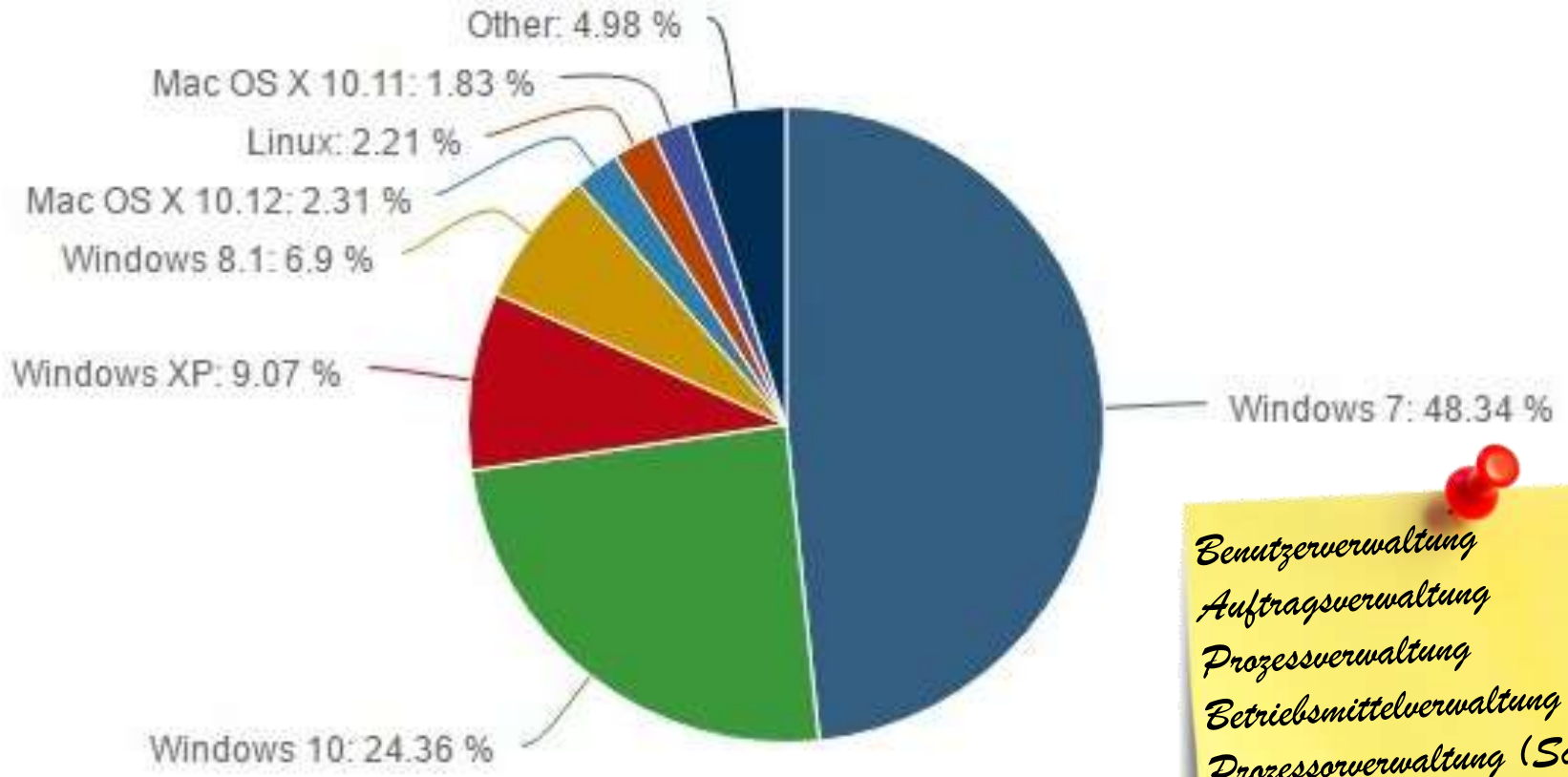
Quelle: KIT IMI

Betriebssystem

- Bindeglied zwischen der Hardware eines Computer
  Anwender bzw. seinen Programmen.



Betriebssysteme (non embedded)

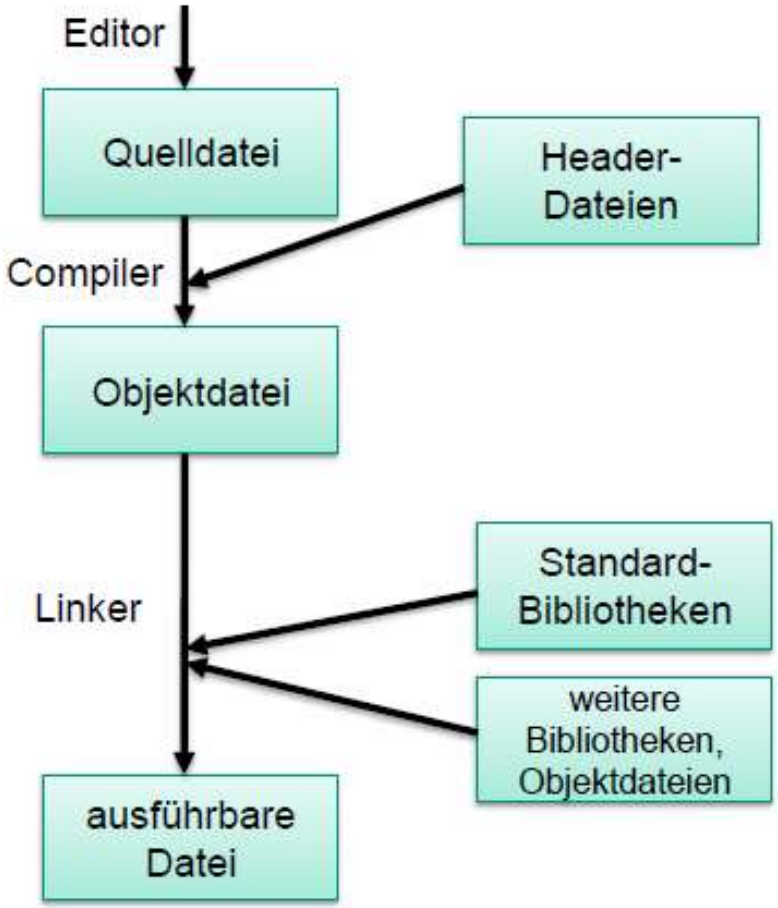


Benutzerverwaltung
Auftragsverwaltung
Prozessverwaltung
Betriebsmittelverwaltung
Prozessorverwaltung (Scheduling)
Hauptspeicherverwaltung
Dateiverwaltung
Ein-/Ausgabe-Steuerung
Kommunikation mit der Umgebung

Stand: Jan. 2017

https://www.pcwelt.de

Erstellen der ausführbaren Anwendung



IT Inhalt

1. **Einleitung und Motivation**
   - Organisatorisches
   - Begriffe
   - Typische Anwendungen
2. **Programmiersprachen**
   - Klassifikationen und Software-Komponenten (aus DT)
   - Was ist eine Programmiersprache?
   - Formale Sprachen
   - Höhere Sprachen
   - Programmierparadigmen
   - Wichtige Programmiersprachen

Programmiersprachen: Formale Sprachen

- Definieren eine bestimmte Menge von Zeichenketten, die aus einem Zeichenvorrat zusammengesetzt werden können
  - Grundlage für höhere Programmiersprachen
  - Voraussetzung für Compilerbau
  - Produktionsregeln und Eindeutigkeit
- Eine Sprache besteht aus einem Alphabet von Zeichen, wobei die Wörter einer Sprache durch (formale) Regeln - der Grammatik - gebildet werden.

Programmiersprachen: Formale Sprachen

- Wörter werden aus gegebenem Alphabet gebildet
  - Bildungsregeln für Wörter in der Grammatik
- Die Syntax
  - einer Programmiersprache beschreibt die Menge der erlaubten Zeichenketten für Programme
- Die Semantik
  - einer Programmiersprache definiert die Bedeutung der einzelnen Sprachkonstrukte (meist textuelle Beschreibung)
- Die Grammatik
  - einer Programmiersprache legt die Produktionsregeln, also den Zusammenhang zwischen den Zeichen und den Wörtern, fest.
  - Formales Beschreibungsmittel: Backus-Naur-Form, Syntaxdiagramme

→ *Formale Sprachen erlauben es, nicht nur Zeichen aus Alphabeten zu Wörtern zusammenzusetzen, sondern Wörter durch Verknüpfung miteinander zu Aussagen.*

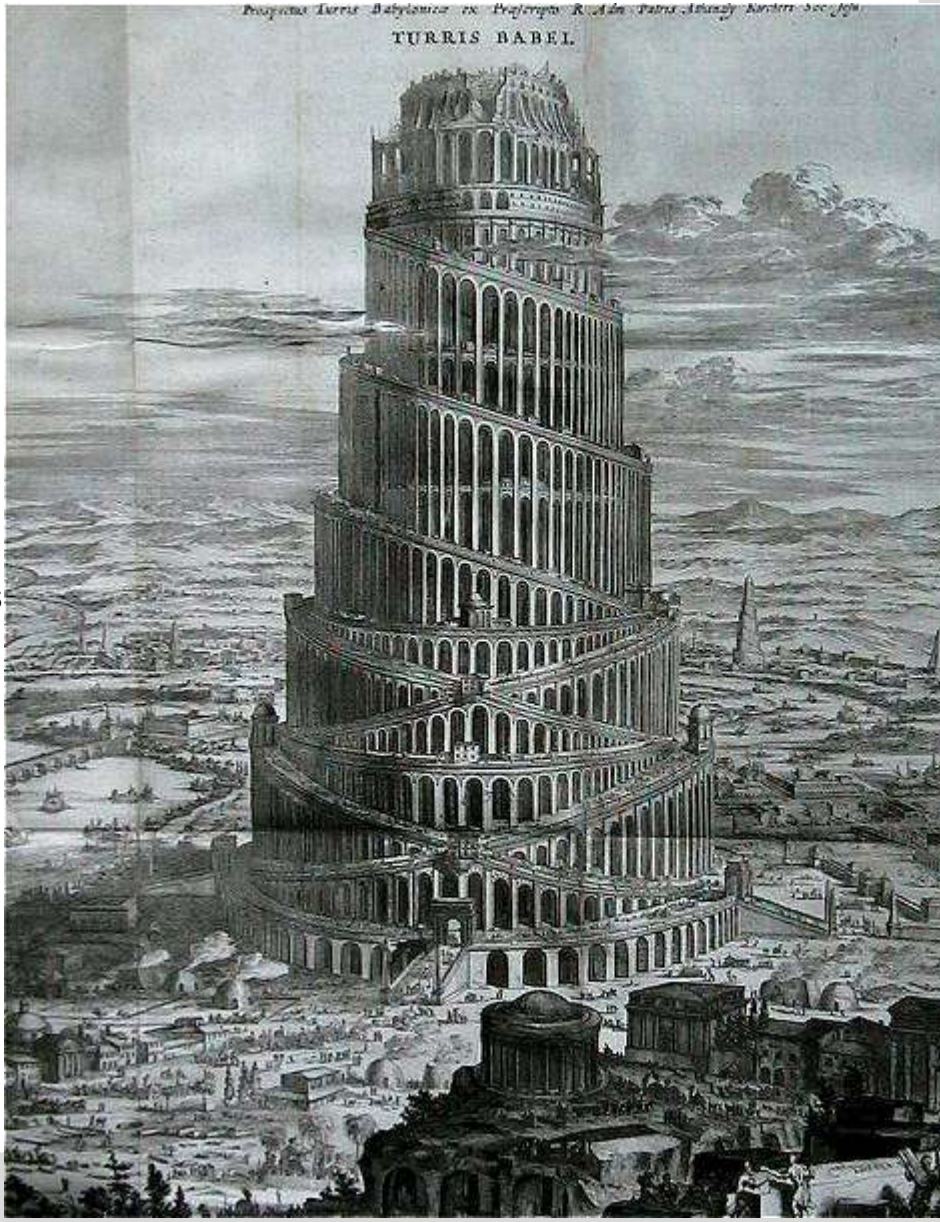
Natürliche Sprache

- Wörter werden aus gegebenem Alphabet gebildet
  - Bildungsregeln für Wörter

Programmiersprachen

1. Mose - Kapitel 11; Der Turmbau zu Babel

- 1 Es hatte aber alle Welt einerlei Zunge und Sprache.
- 2 Da sie nun zogen gen Morgen, fanden sie ein ebenes Land im Lande Sinear, und wohnten daselbst.
- 3 Und sie sprachen untereinander: Wohlauf, laß uns Ziegel streichen und brennen! und nahmen Ziegel zu Stein und Erdharz zu Kalk
- 4 und sprachen: Wohlauf, laßt uns eine Stadt und einen Turm bauen, des Spitze bis an den Himmel reiche, daß wir uns einen Namen machen! denn wir werden sonst zerstreut in alle Länder.
- 5 Da fuhr der HERR hernieder, daß er sähe die Stadt und den Turm, die die Menschenkinder bauten. (1. Mose 18.21) (Psalm 14.2) (Psalm 18.10)
- 6 Und der HERR sprach: Siehe, es ist einerlei Volk und einerlei Sprache unter ihnen allen, und haben das angefangen zu tun; sie werden nicht ablassen von allem, was sie sich vorgenommen haben zu tun.
- 7 Wohlauf, laßt uns herniederfahren und ihre Sprache daselbst verwirren, daß keiner des andern Sprache verstehe!
- 8 Also zerstreute sie der HERR von dort alle Länder, daß sie mußten aufhören die Stadt zu bauen. (Lukas 1.51)
- 9 Daher heißt ihr Name Babel, daß der HERR daselbst verwirrt hatte aller Länder Sprache und sie zerstreut von dort in alle Länder.



Programmiersprachen

1. Mose - Kapitel 11; Der Turmbau zu Babel

1 Es hatte aber alle Welt einerlei Zunge und Sprache.

2 Da sie nun zogen gen Morgen, fanden sie ein ebenes Land im Lande Sinear, und wohnten daselbst.

3 Und sie sprachen untereinander: Wohlauf, laß uns Ziegel streichen und brennen! und nahmen Ziegel zu Stein und Erdharz zu Kalk

4 und sprachen: Wohlauf, laßt uns eine Stadt und einen Turm bauen, des Spitze bis an den Himmel reiche, daß wir uns einen Namen machen! denn wir werden sonst zerstreut in alle Länder.

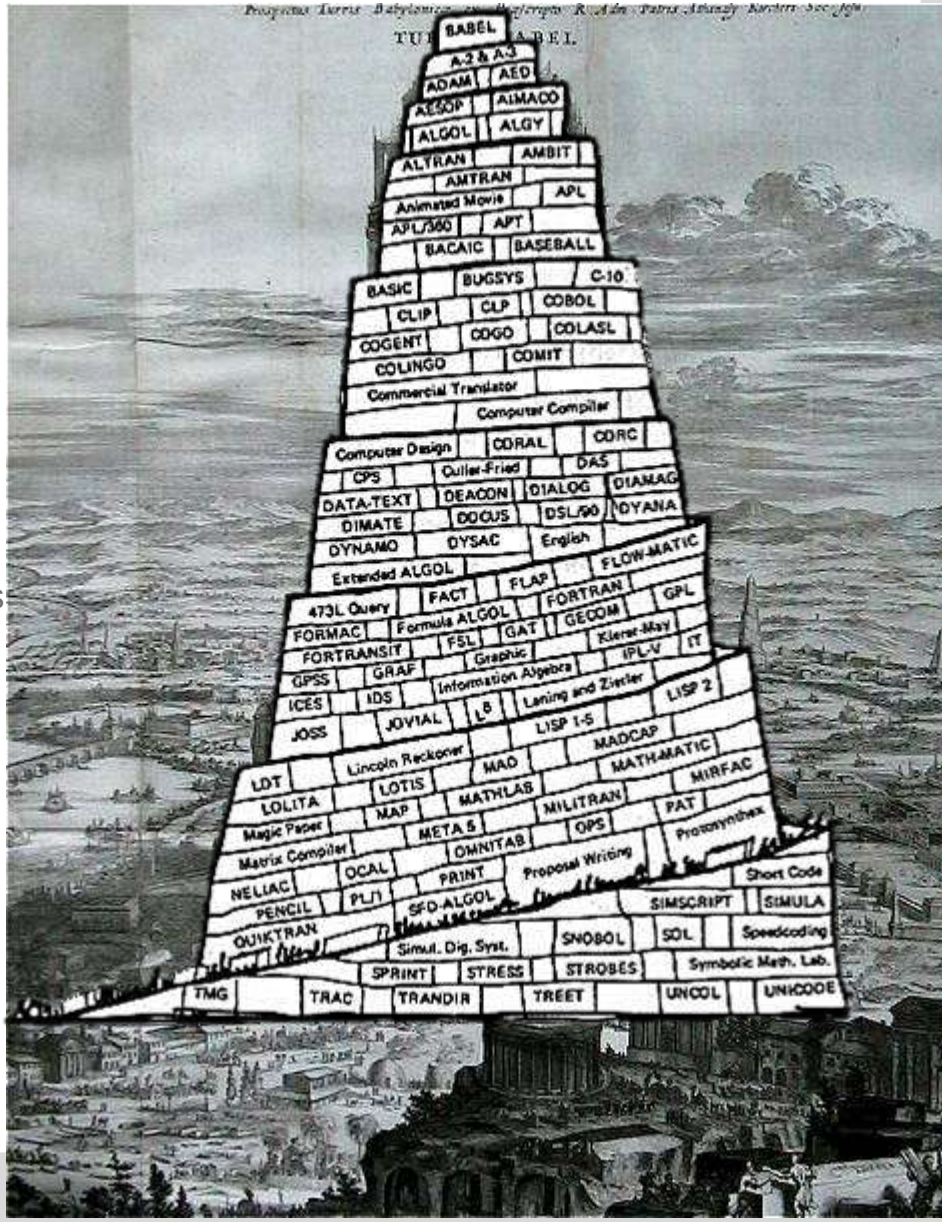
5 Da fuhr der HERR hernieder, daß er sähe die Stadt und den Turm, die die Menschenkinder bauten. (1. Mose 18.21) (Psalm 14.2) (Psalm 18.10)

6 Und der HERR sprach: Siehe, es ist einerlei Volk und einerlei Sprache unter ihnen allen, und haben das angefangen zu tun; sie werden nicht ablassen von allem, was sie sich vorgenommen haben zu tun.

7 Wohlauf, laßt uns herniederfahren und ihre Sprache daselbst verwirren, daß keiner des andern Sprache verstehe!

8 Also zerstreute sie der HERR von dort alle Länder, daß sie mußten aufhören die Stadt zu bauen. (Lukas 1.51)

9 Daher heißt ihr Name Babel, daß der HERR daselbst verwirrt hatte aller Länder Sprache und sie zerstreut von dort in alle Länder.



Programmiersprachen

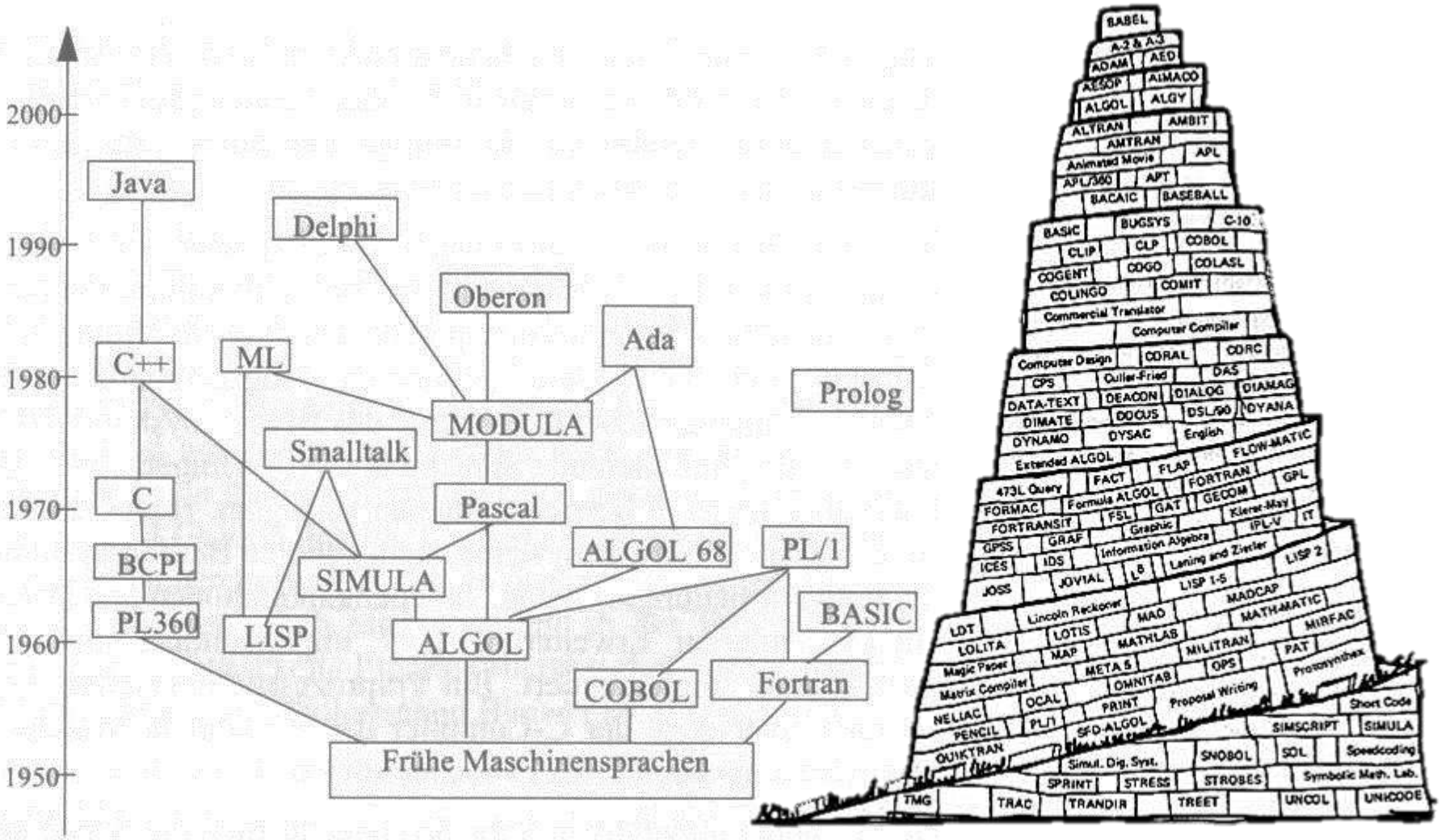
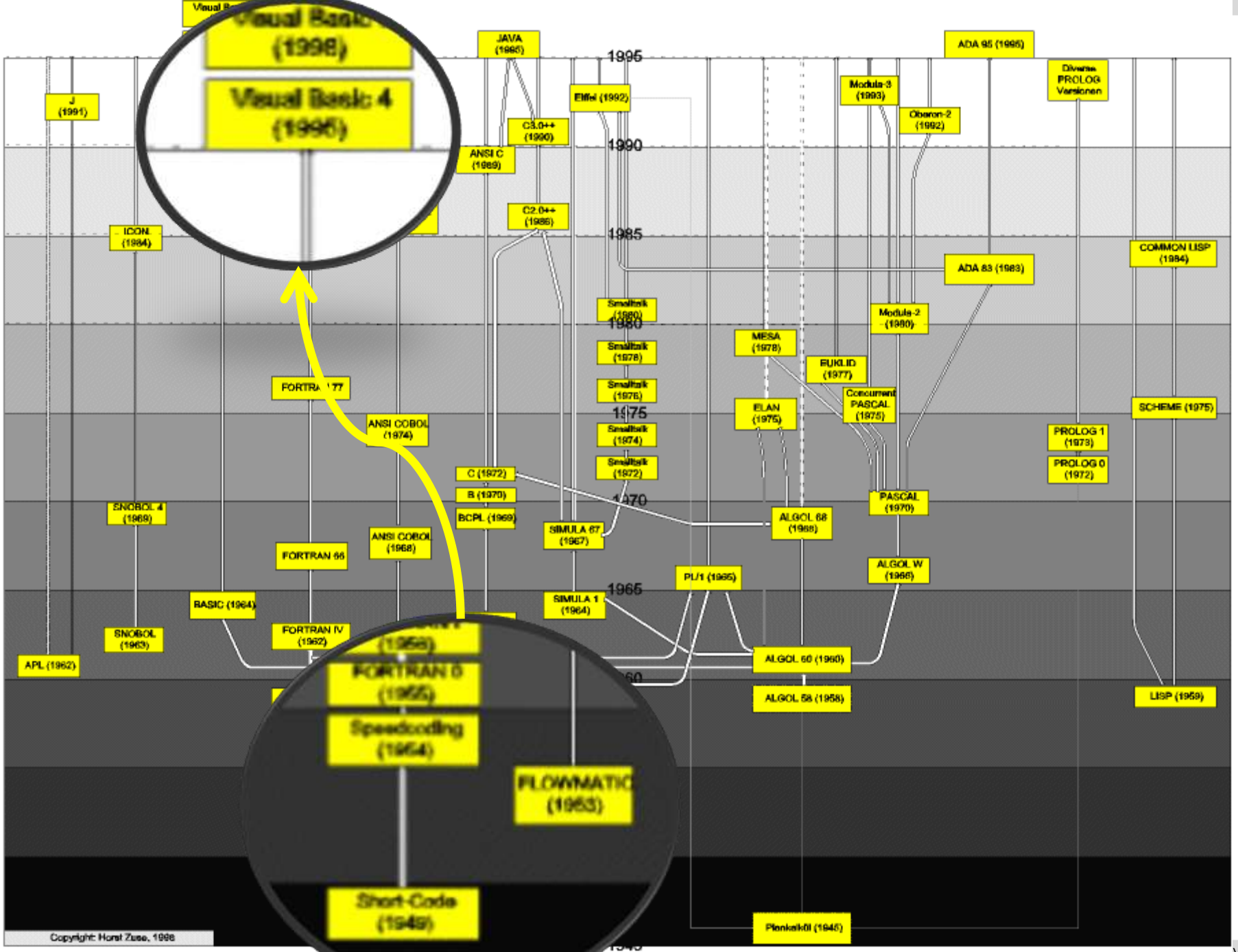


Abb. 9-1: Entstehung der Programmiersprachen

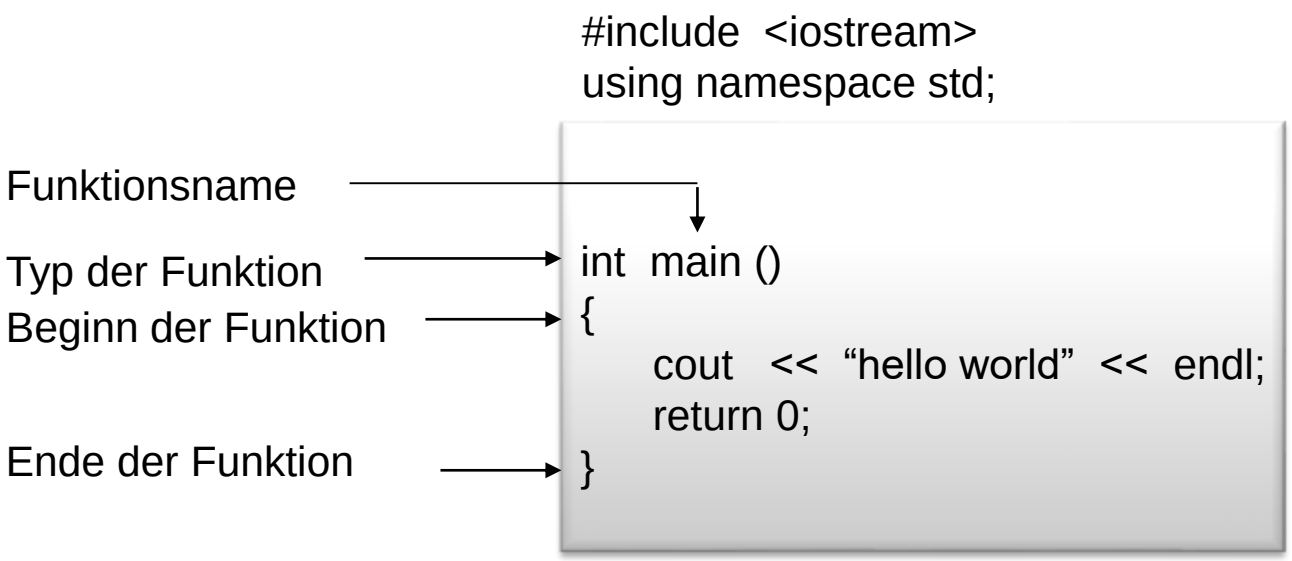

Visual Basic (1998)
Visual Basic 4 (1995)
J (1991)
JAVA (1995)
Eiffel (1992)
C3.0++ (1990)
ANSI C (1989)
C2.0++ (1985)
ICON (1984)
ADA 95 (1995)
Modula-3 (1993)
Oberon-2 (1992)
Diverse PROLOG Versionen
COMMON LISP (1984)
ADA 83 (1983)
Smalltalk (1980)
Modula-2 (1980)
MESA (1978)
EUKLID (1977)
Smalltalk (1978)
Smalltalk (1976)
Concurrent PASCAL (1975)
ELAN (1975)
SCHEME (1975)
FORTRAN 77
ANSI COBOL (1974)
Smalltalk (1974)
Smalltalk (1972)
PROLOG 1 (1973)
PROLOG 0 (1972)
C (1972)
B (1970)
BCPL (1969)
PASCAL (1970)
SNOBOL 4 (1969)
SIMULA 67 (1967)
ALGOL 68 (1968)
ANSI COBOL (1968)
FORTRAN 66
ALGOL W (1966)
PL/1 (1965)
BASIC (1964)
SIMULA 1 (1964)
SNOBOL (1963)
FORTRAN IV (1962)
APL (1962)
ALGOL 60 (1960)
ALGOL 58 (1958)
LISP (1959)
FORTRAN II (1958)
FORTRAN 0 (1955)
Speedcoding (1954)
FLOWMATIC (1953)
Short-Code (1949)
Plankalkül (1945)
1995
1990
1985
1980
1975
1970
1965
1960
1945
Copyright: Horst Zuse, 1998

Programmierparadigmen

Paradigma	im Vordergrund stehen	Beispiel
Imperativ	**Variablen**, die einzelnen oder mehreren Speicherzellen des Rechners entsprechen, und **Anweisungen („Befehle“)** als Abstraktionen der Prozessor-Funktionen	Assembler-Sprachen, FORTRAN, ALGOL
Prozedural	**Prozeduren** (Unterprogramme), an denen **Argumente (Parameter)** übergeben werden. Die Prozeduren berechnen **Werte** nach einem **Algorithmus** und geben diese zurück	Pascal, C
Funktional	**(Mathematische) Funktionen**, die einem **Vektor von Parametern** (darunter evtl. auch Funktionen) einen **Wert** zuordnen	LISP, LOGO HASKELL
Logikbasiert	**Logische Aussagen,** die im allgemeinen **freie Variablen** enthalten und von denen geprüft wird, ob und ggf. wie sie sich durch eine geeignete Bindung dieser Variablen **verifizieren** lassen	PROLOG
Objektorientiert	Autonome, interagierende **Objekte,** die durch **Botschaften** kommunizieren und mit anderen, ähnlichen Objekten in **Klassen** zusammengefasst sind	Smalltalk, C++, Java, C#

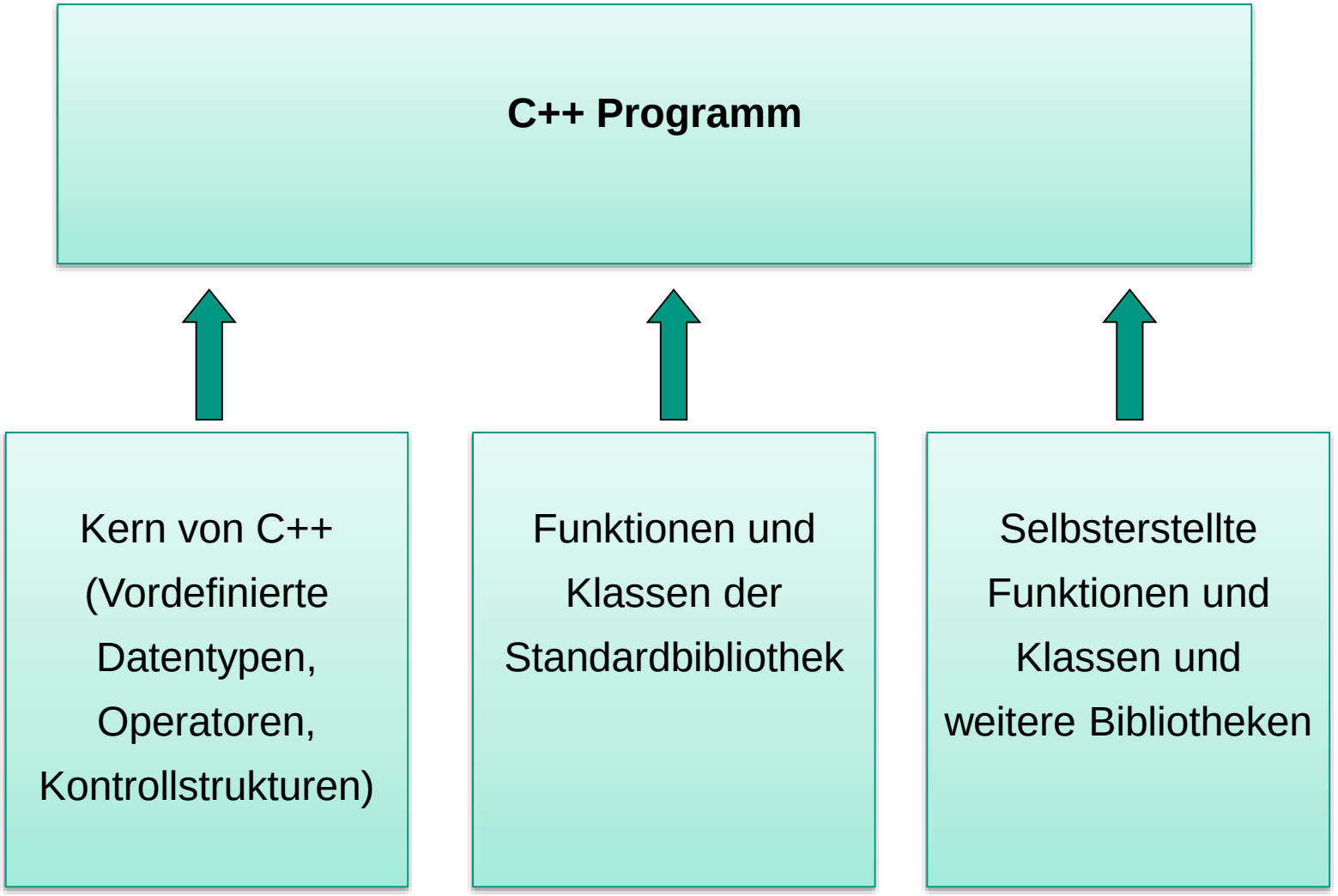
Quelle: [ApLu99]

Formale Sprachen: Ein erstes C++-Programm



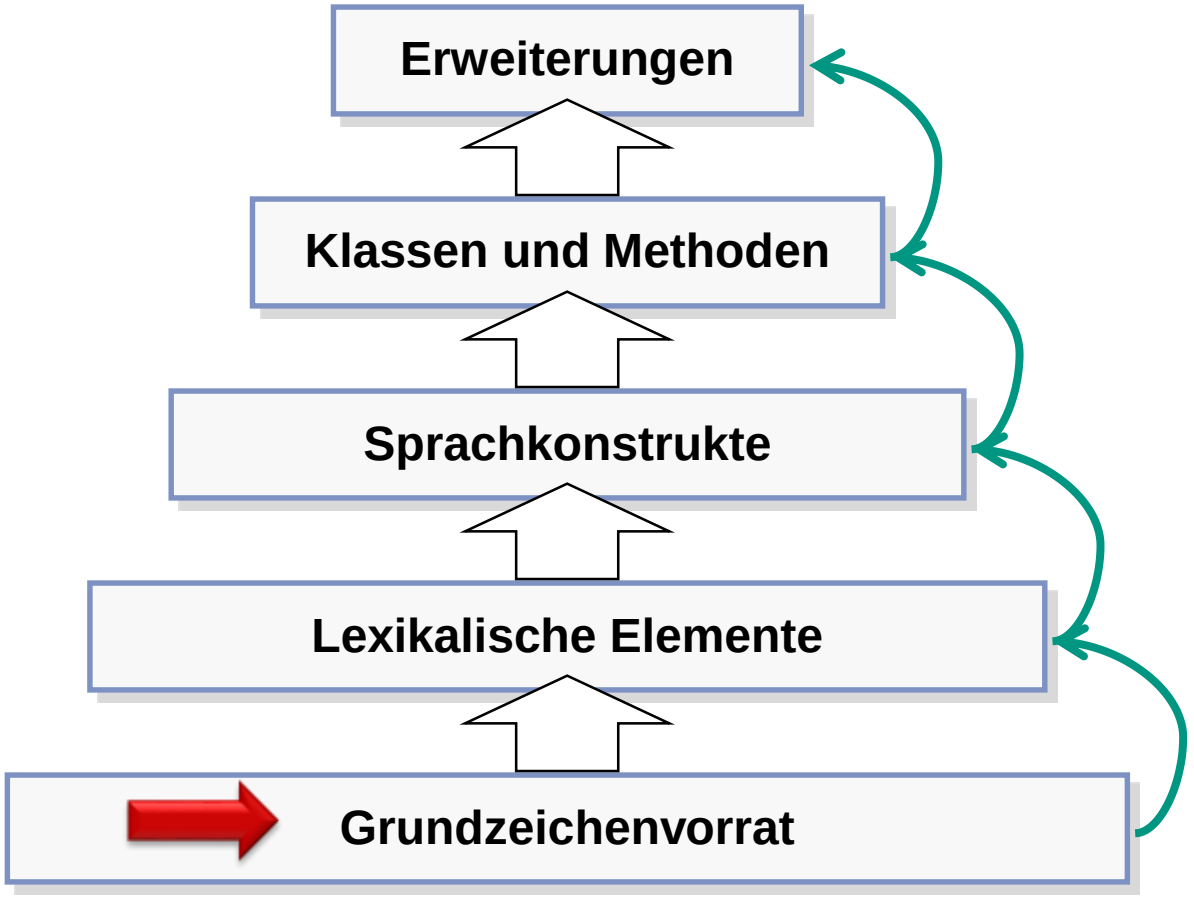
- Jedes C++ Programm startet mit der Abarbeitung der Funktion mit dem Namen „main“.
- Alle anderen Funktionen werden dadurch definiert, dass sie nacheinander aufgeführt werden.
- Funktionen die aufgerufen werden, müssen vorher definiert sein.
- Die Definition von Funktionen in Funktionen ist nicht erlaubt

Bestandteile eines C++ Programms



Kern von C++ / Sprachaufbau

Kern von C++
(Vordefinierte Datentypen, Operatoren, Kontrollstrukturen)



Alphabet: 7-bit ASCII-Tabelle

American Standard Code for Information Interchange (ASCII) ist eine 7-Bit-Zeichenkodierung; sie entspricht der US-Variante von ISO 646 und dient als Grundlage für spätere auf mehr Bits basierenden Kodierungen für Zeichensätze.

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1-	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2-	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

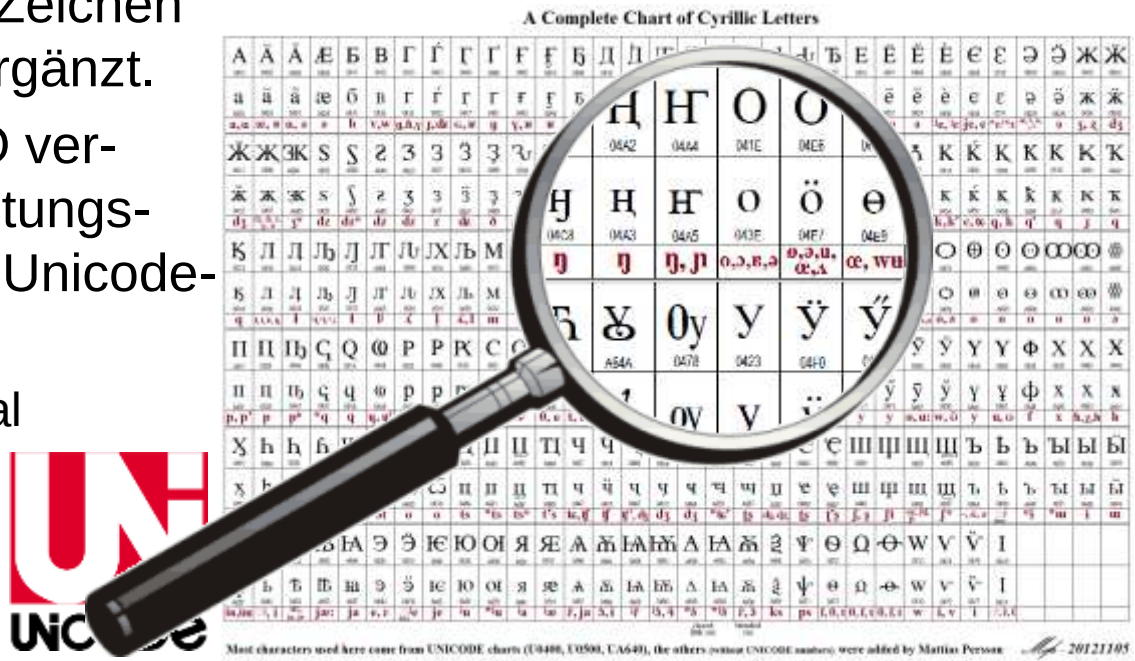
ASCII Kodierung und Unicode

- Kodieren Sie folgende Zeichenkette mit dem ASCII Code in hexadezimaler Darstellung:
  - Hello IT!
  - Lösung: 48 65 6C 6C 6F 20 49 54 21

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1-	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2-	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

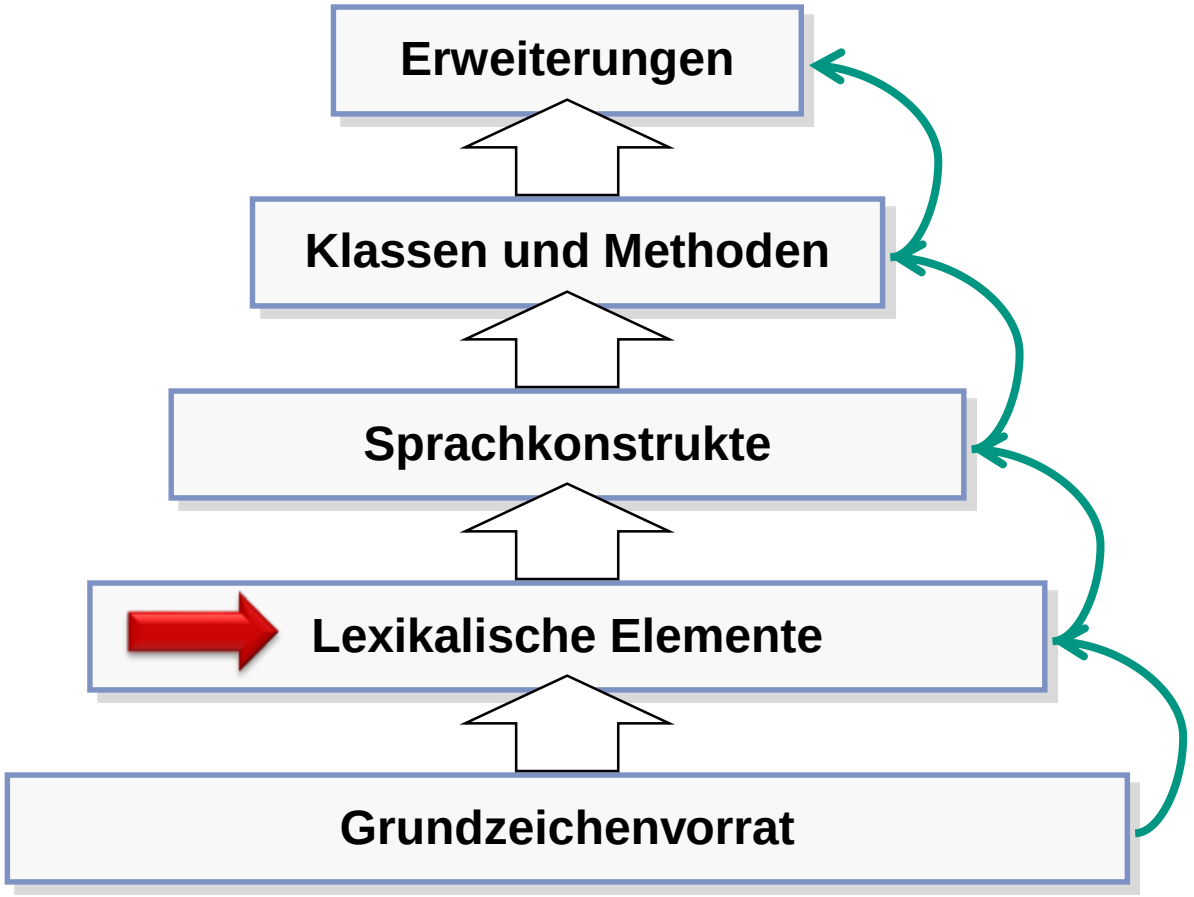
Unicode

- Internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird.
- Ziel ist, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen.
- Unicode wird ständig um Zeichen weiterer Schriftsysteme ergänzt.
- ISO 10646 ist die von ISO verwendete, praktisch bedeutungsgleiche Bezeichnung des Unicode-Zeichensatzen
  - er wird dort als Universal Character Set (UCS) bezeichnet.



Kern von C++ / Sprachaufbau

Kern von C++
(Vordefinierte
Datentypen,
Operatoren,
Kontrollstrukturen)



Lexikalische Elemente: Kommentare

- Syntax :

```
// beliebiger Text aus dem Zeichensatzvorrat von C++
```

- Beispiele :

```
// Ein Kommentar beginnt mit // und reicht
// bis zum Zeilenende.
// Er kann alle moeglichen Zeichen (*'_<->) beinhalten
// und über mehrere Zeilen gehen, wenn er mit /* … */
// begrenzt wird
```

- Zur Verdeutlichung des Programms (menschenlesbar)
- Meist bedeutungslos für Compiler

Lex. El.: Trenn- und Begrenzungszeichen (1)

- Dienen zur Trennung aufeinanderfolgender lexikalischer Elemente
- Einzelzeichen (neben Leerzeichen) :

```
(  )  [  ]  {  }  |  &  !  #  .  ,  :  ;  /  %  *  -  <  =  >  +
```

- Zusammengesetzte Zeichen :

```
>=  <=  ==  !=  &&  ||  ++  --  +=  -=  *=  /=
```

(Aufzählung ist unvollständig)

Lex. El.: Trenn- und Begrenzungszeichen (2)

- Befehlskette ohne Trenn- und Begrenzungszeichen:

```
inta=5b=7ifa>=ba-belseb-a
```

  - → Funktion nicht interpretierbar
- Befehlskette mit Trenn- und Begrenzungszeichen:

```
int a=5, b=7; if(a>=b){a-b;} else{b-a;}
```

  - → Betragsberechnung der Differenz
  - ABER: Kein guter Programmierstil!
  - → Strukturierung durch Einrücken →

```
int a=5, b=7;
   if(a>=b) {
      a-b;
      }
   else {
      b-a;
      }
```

Lexikalische Elemente: Bezeichner („Identifier“)

- Namen von Variablen, Kostanten, Objekten, Typen usw.
- Bezeichner bestehen aus Zeichen des 7-bit ASCII Zeichensatzes
  - Buchstaben
  - Ziffern
  - Einzelnen Unterstrichen
  - Keine Leer- und weitere Sonderzeichen erlaubt!
- Bezeichner sind „case-sensitive“
- Erstes Zeichen muss ein Buchstabe oder ein Unterstrich sein
- Unterstrich (“_”) darf nicht …
  - … zweimal unmittelbar aufeinanderfolgen (.Net Schlüsselworterweiterungen __abstract, __event, …)
  - … am Ende eines Bezeichners stehen
- Bezeichner dürfen keine reservierten Worte (s. „Schlüsselworte“ nächste Folie) sein

Lex. El.: Schlüsselworte in C++

asm	do	inline	short	typeid
auto	double	int	signed	typename
bool	dynamic_cast	long	sizeof	unsigned
break	else	mutable	static	virtual
case	enum	namespace	static_cast	void
catch	explicit	new	struct	volatile
char	extern	operator	switch	wchar_t
class	false	private	template	while
const	float	protected	this	
const_cast	for	public	throw	
continue	friend	register	true	
default	goto	reinterpret_cast	try	
delete	if	return	typedef	

Lex. El.: Elementare Datentypen (Literale, Größen)

- numerische Größen
- Zeichen (character), Zeichenketten (strings), „Bit-Strings“

Character

'A'
'a'
'@'
'"'
'1'
'+'
'x'

Strings

"string \"IN\" a string"
"This is a string"
"xy"
""

Integerzahlen

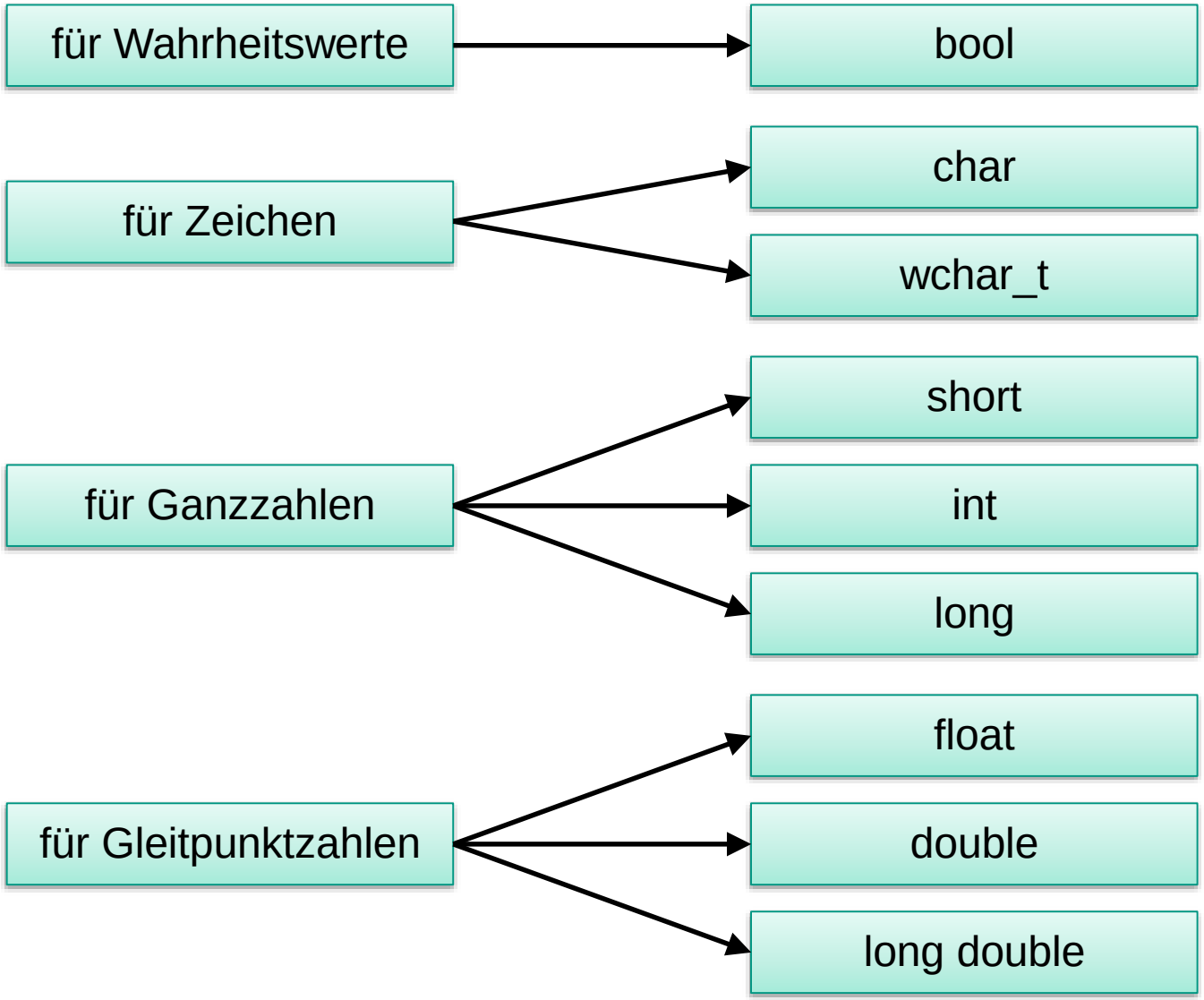
2
3E4
100000
2153E3

Fließkommazahlen

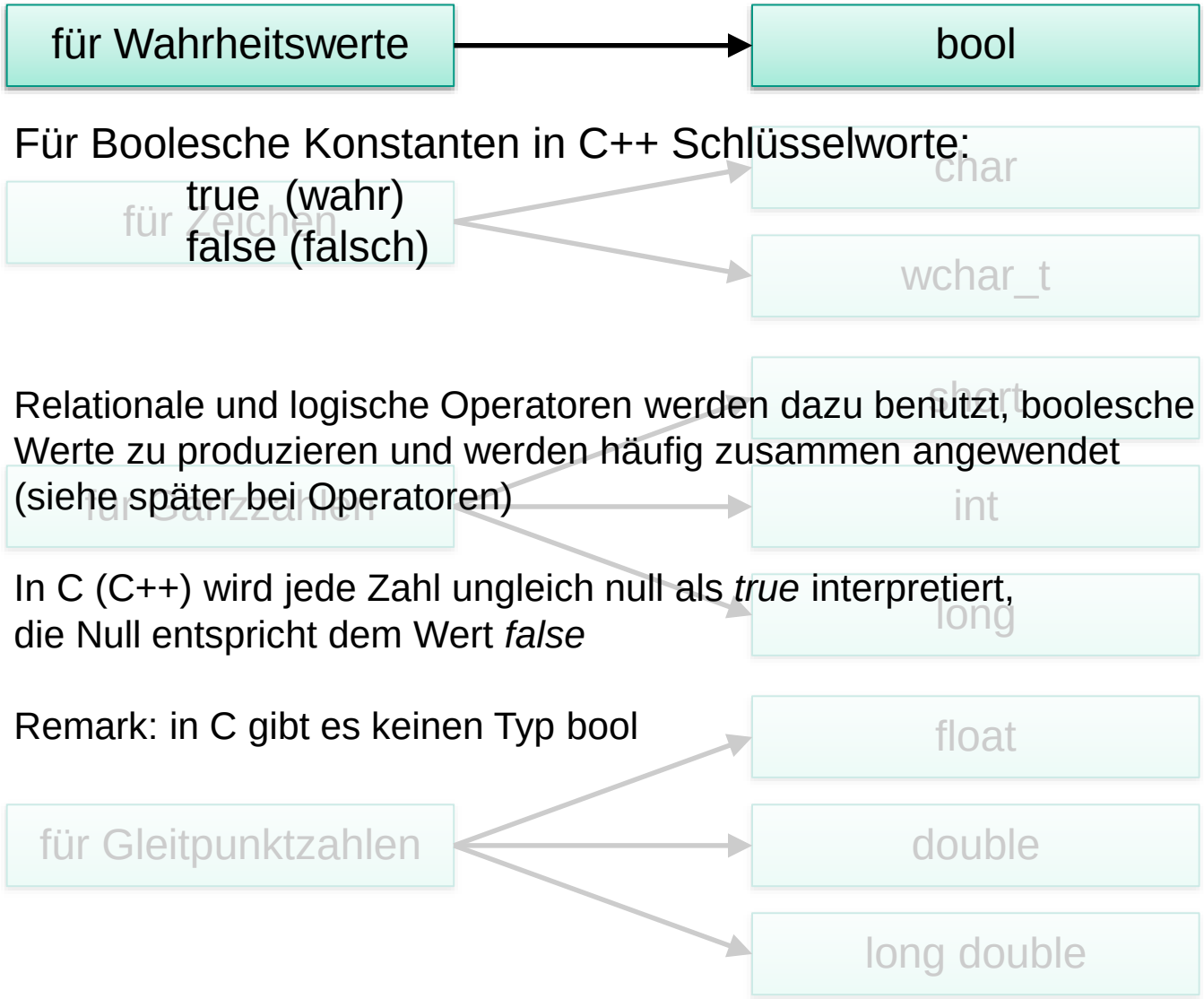
2.3
0.04
1.4E12
1.00345E-6

- Compiler muss erkennen (Syntax, Semantik), um welchen Datentyp es sich handelt

Lex. El.: Elementare Datentypen



Lex. El.: Elementare Datentypen

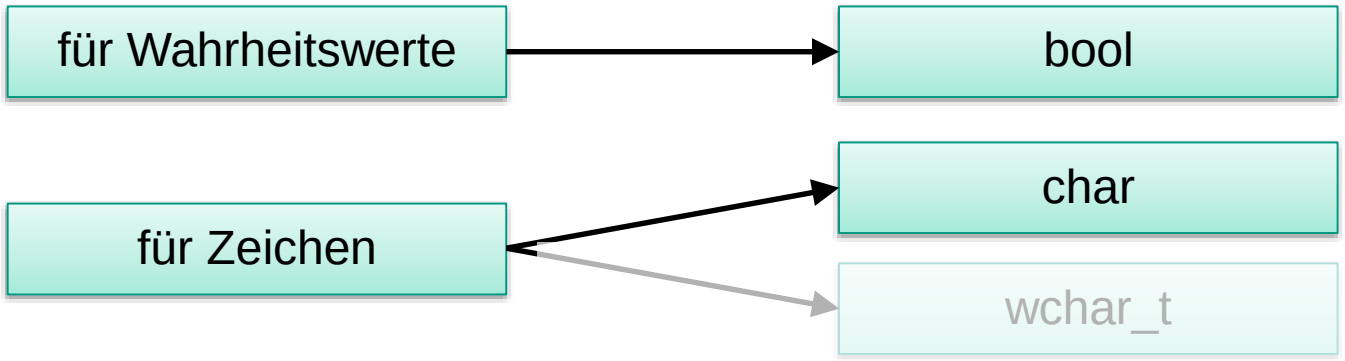


Relationale und logische Operatoren werden dazu benutzt, boolesche Werte zu produzieren und werden häufig zusammen angewendet (siehe später bei Operatoren)

In C (C++) wird jede Zahl ungleich null als *true* interpretiert, die Null entspricht dem Wert *false*

Remark: in C gibt es keinen Typ bool

Lex. El.: Elementare Datentypen

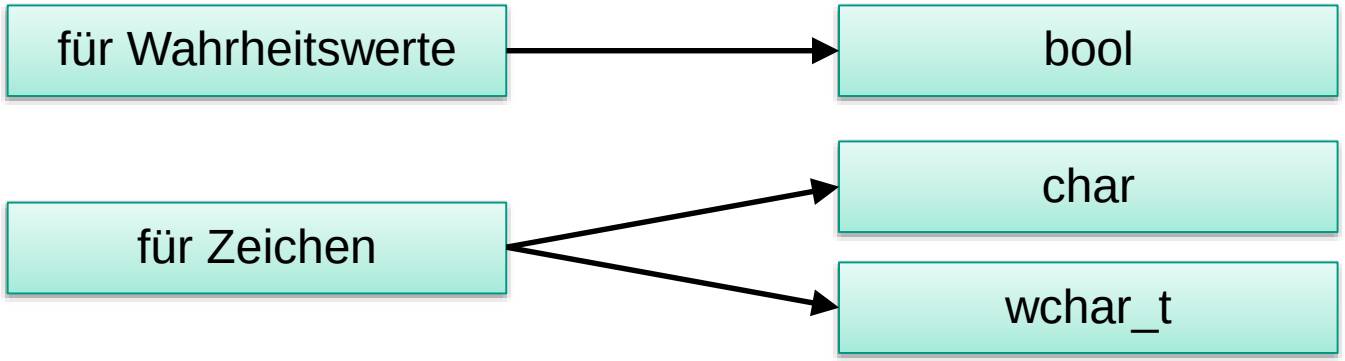


- dienen zur Speicherung von Zeichencodes (1 Byte)
- jedem Zeichen ist eine ganze Zahl zugeordnet
- Zuordnung wird festgelegt durch verwendeten Zeichensatz
  - C++ legt keinen Zeichensatz fest
  - üblich 7-bit ASCII Code
    - Codes 0 bis 31 Steuerzeichen
    - 32 bis 127 druckbare Zeichen

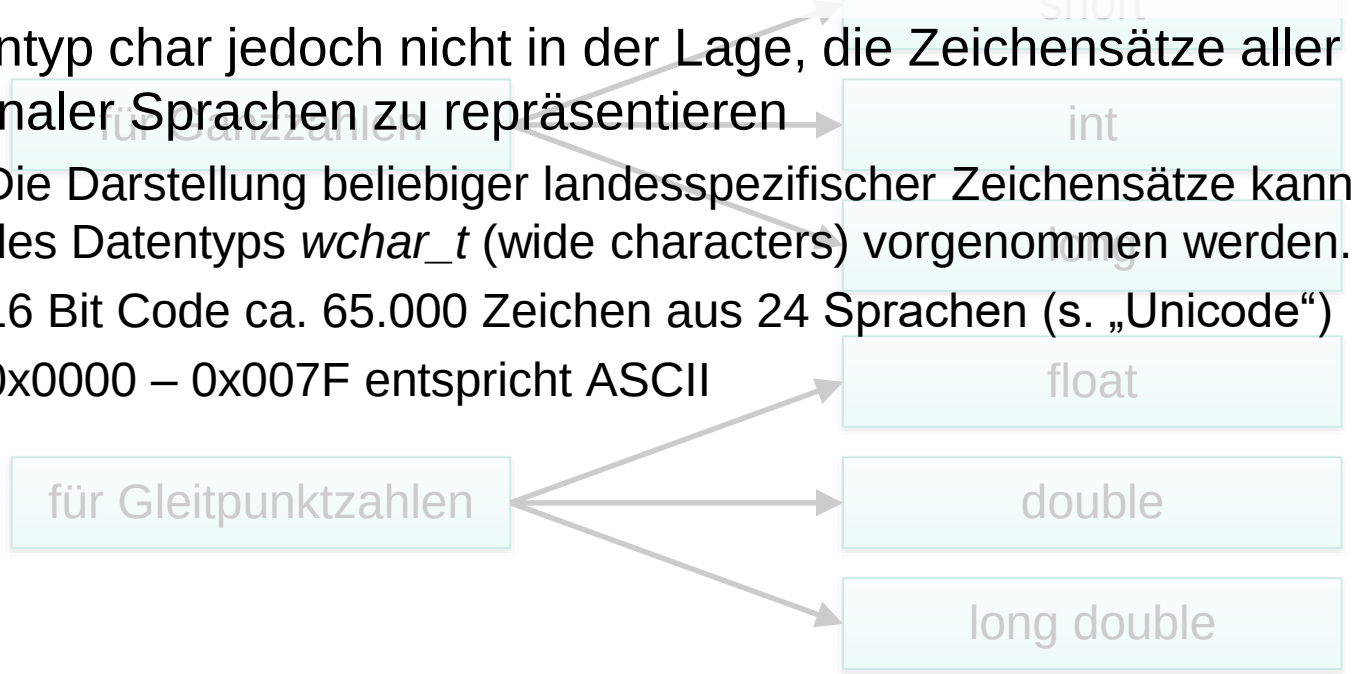
für Gleitpunktzahlen

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1-	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2-	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Lex. El.: Elementare Datentypen

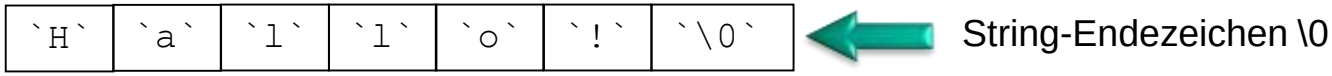


- Char → erweiterter Zeichensatz ANSI 8-bit Code (z.B. für Umlaute)
- Datentyp char jedoch nicht in der Lage, die Zeichensätze aller nationaler Sprachen zu repräsentieren
  - Die Darstellung beliebiger landesspezifischer Zeichensätze kann mit Hilfe des Datentyps *wchar_t* (wide characters) vorgenommen werden.
  - 16 Bit Code ca. 65.000 Zeichen aus 24 Sprachen (s. „Unicode“)
  - 0x0000 – 0x007F entspricht ASCII

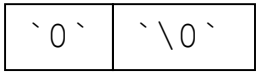


Beispiele für Zeichenketten (string Konstante)

- String-Konstante: Im Speicher abgelegt als Byte-Folge
- “Hallo!“



- “0“



- Escape Sequenzen finden in strings
  Verwendung für grafisch nicht darstell-bare Zeichen:

```
cout << "Programming is\n";
cout << "fun!";
```

Typische Escape-Sequenzen:

Einzel-zeichen	Bedeutung	ASCII-Wert (dezimal)
\a	alert (BEL)	7
\b	backspace (BS)	8
\t	horizontal tab (HT)	9
\n	line feed (LF)	10
\v	vertikal tab (VT)	11
\f	form feed (FF)	12
\r	carriage return (CR)	13
\“	“	34
\\	\	92
\0	Stringende-Zeichen	0

Beispiele für Zeichenketten (string Konstante)

- String-Konstante: Im Speicher abgelegt als Byte-Folge
- “Hallo!“

`H`	`a`	`l`	`l`	`o`	`!`	`\0`

String-Endezeichen \0

- “0“

`0`	`\0`

- Escape Sequenzen finden in strings
  Verwendung für grafisch nicht darstell-bare Zeichen:

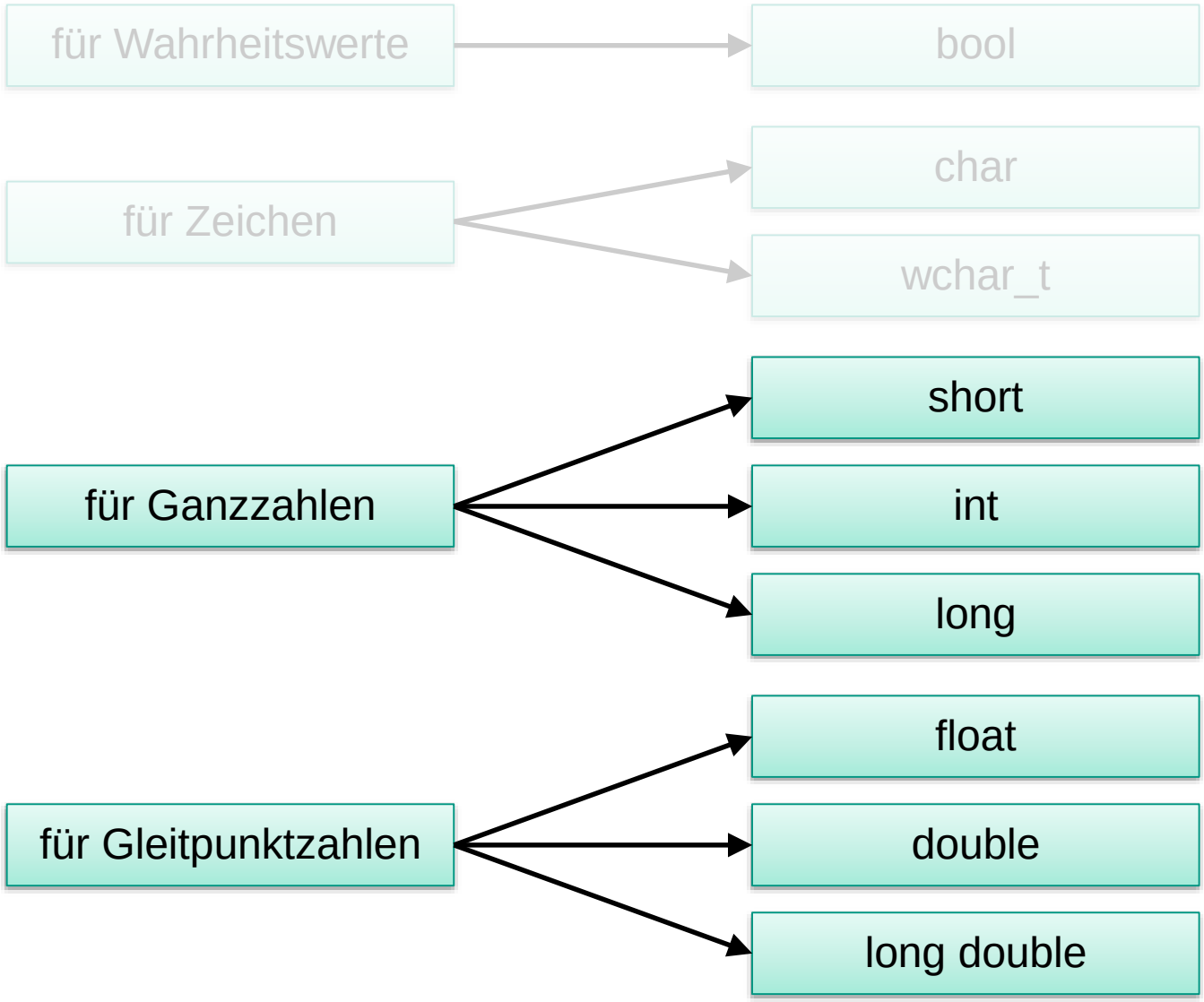
```
cout << "Programming is\n";
cout << "fun!";
```

Typische Escape-Sequenzen

Einzel-zeichen	Bedeutung	ASCII-Wert (dezimal)
\a	alert (BEL)	7
\b	backspace (BS)	8
\t	horizontal tab (HT)	9
\n	line feed (LF)	10
\v	vertikal tab (VT)	11
\f	form feed (FF)	12
	rn (CR)	13
		34
		92
	eichen	0

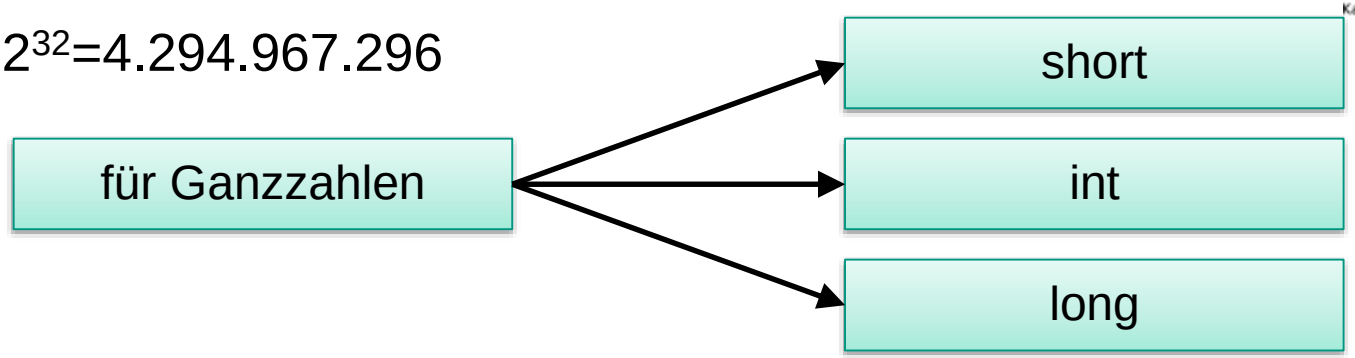
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1-	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US

Beispiele für Zeichenkonstanten (character)



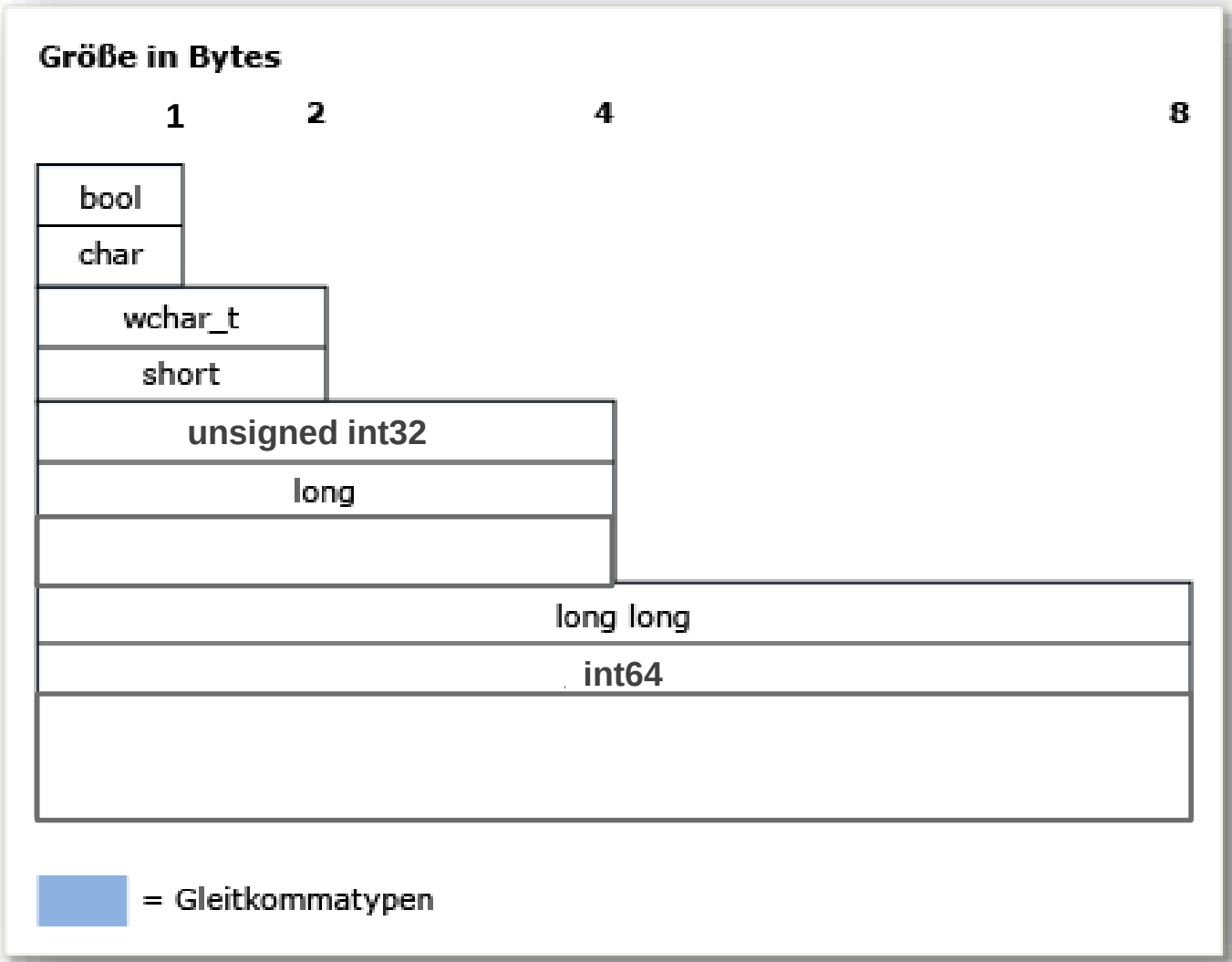
Die ganzzahligen Datentypen

4 Byte: 2^{32}=4.294.967.296

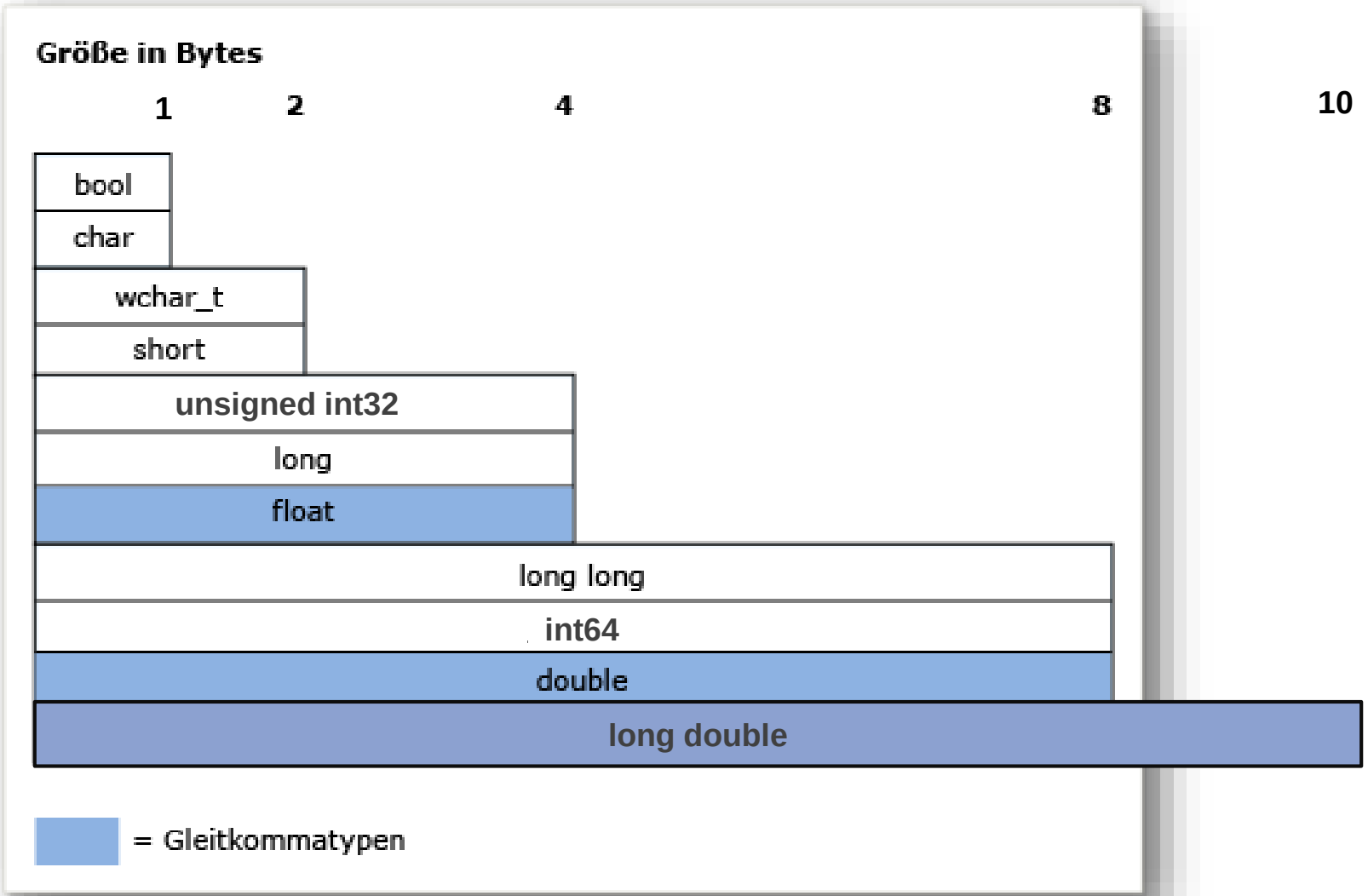


Typ	Speicherplatz	Wertebereich
`char`	1 Byte	-128 bis +127 bzw. 0 bis 255
`unsigned char`	1 Byte	0 bis 255
`signed char`	1 Byte	-128 bis +127
`int16 (short)`	2 Byte bzw.	-32.768 bis +32.767 bzw.
`int32 (long)`	4 Byte	-2.147.483.648 bis +2.147.483.647
`unsigned int16`	2 Byte bzw.	0 bis 65.535 bzw.
`unsigned int32`	4 Byte	0 bis 4.294.967.295

Größe der integrierten Datentypen



Größe der integrierten Datentypen

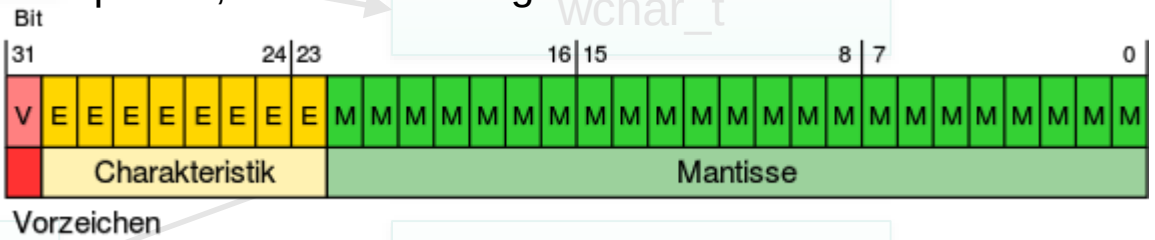


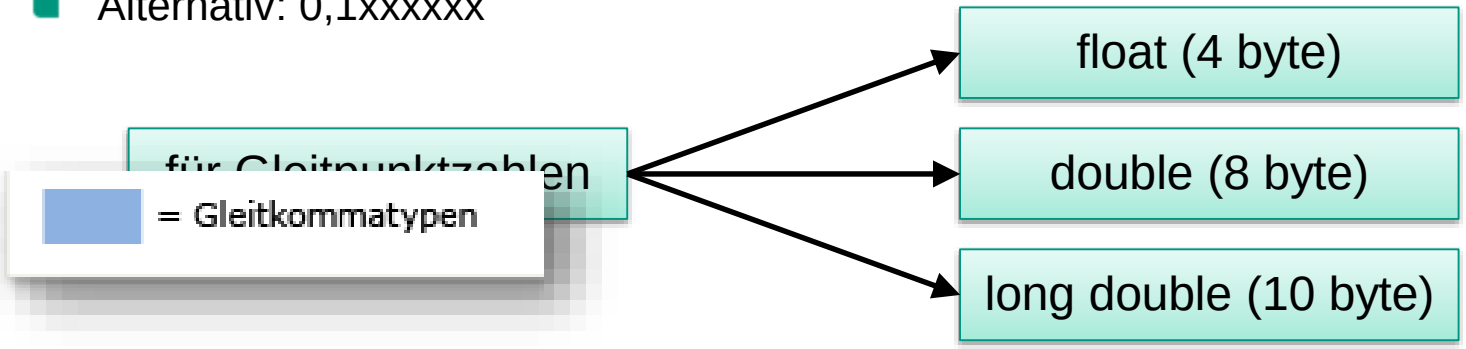
Die Datentypen für Gleitpunktzahlen

ANSI/IEEE Std 754-1985; IEC-60559:1989

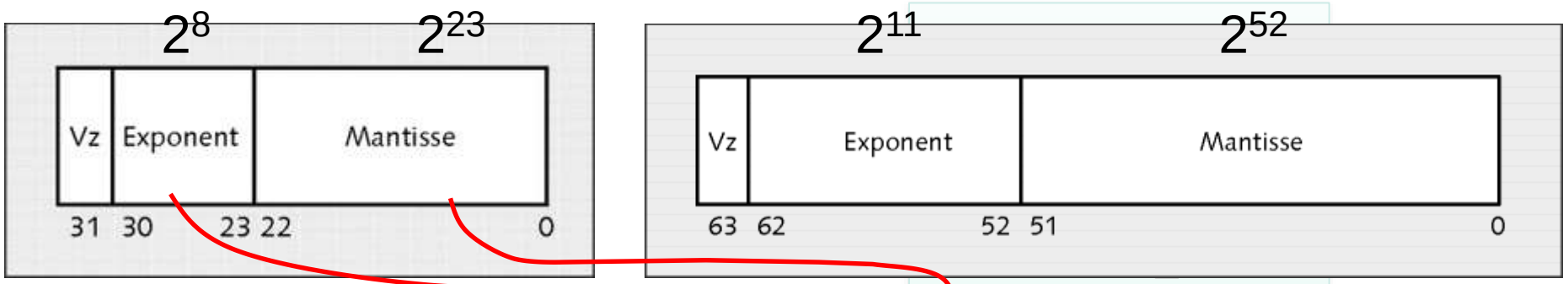
- Vorzeichen-(Vz-)Bit (1 Bit) s:
  - In Bit 31 wird das Vorzeichen der Zahl gespeichert.
  - Ist dieses 0, dann ist die Zahl positiv, bei 1 ist sie negativ.
- Basis b
  - Immer 2 → binär!
- Exponent (8 Bits) e:
  - In Bit 23 bis 30 wird der Exponent gespeichert.
- Mantisse (23 Bits) m:
  - In Bit 0 bis 22 wird der Bruchteil der Mantisse gespeichert.
  - Die Mantisse wird so „normalisiert“, dass der Wert vor dem Komma 1 ist
    - Darstellung 1, xxxxxxxx → $m_1 \times 2^{(-1)} + m_2 \times 2^{(-2)} + ... + m_n \times 2^{(-n)}$
  - Alternativ: 0,1xxxxxx

$$X = Vz \cdot m \cdot b^e$$





Die Datentypen für Gleitpunktzahlen



Typ	Größe (1+r+p)	Exponent (r)	Mantisse (p)	Werte des Exponenten (e)	Biaswert (B)
single	32 bit	8 bit	23 bit	$-126 \leq e \leq 127$	127
double	64 bit	11 bit	52 bit	$-1022 \leq e \leq 1023$	1023

long

Für Sonderfälle stehen spezielle Bitmuster zur Verfügung. Um diese Sonderfälle zu kodieren, sind zwei Exponentenwerte, der maximale (11111…) und die Null (000000…) reserviert.

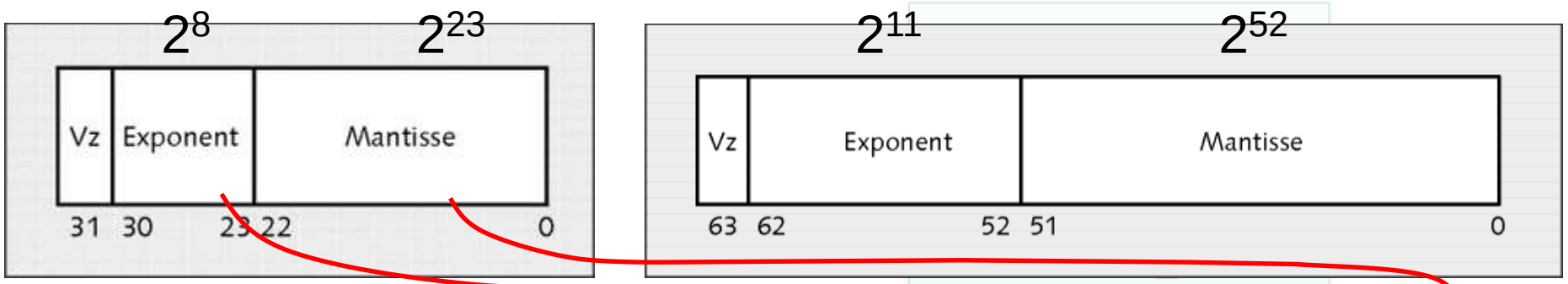
- Mit dem maximalen Exponentenwert werden die Sonderfälle NaN (not a number) bzw. ∞ kodiert.
- Mit Null im Exponenten wird die Gleitkommazahl 0 Werte kodiert.

float

double

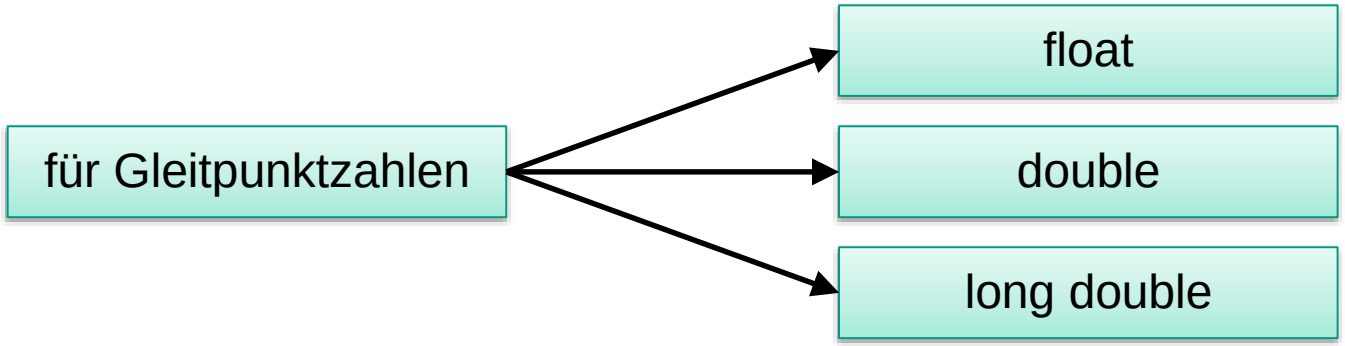
long double

Die Datentypen für Gleitpunktzahlen



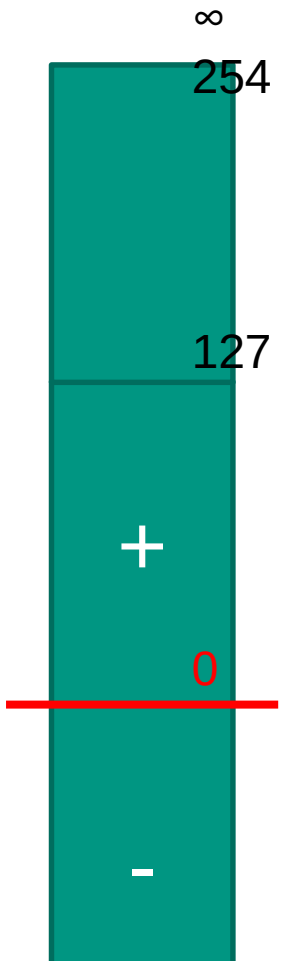
Typ	Speicherplatz	Größte Zahl	kleinste positive Zahl	Genauigkeit (dezimal)
float	4 Bytes	3.4E+38 ≈ $2*2^{127}$	1.2E-38 ≈ 2^{-126}	7 Stellen (2^{23})
double	8 Bytes	1.7E+308 ≈ $2*2^{1023}$	2.3E-308 ≈ 2^{-1022}	16 Stellen (2^{52})
long double	10 Bytes	1.1E+4932	3.4E-4932	19 Stellen

Remark: Die Mantisse ist rechnerisch durch das implizite Bit um noch eins größer als gespeichert.



Darstellung des Exponenten-Vorzeichens mit Bias

- In Gleitkommasystemen ist der Exponent eine Zahl mit Vorzeichen.
- Würde Implementierung einer zusätzlichen Arithmetik mit Vorzeichen für Exponenten-Berechnungen erforderlich machen.
  - kann vermieden werden, wenn zum Exponenten eine feste Zahl, (*Biaswert* oder *Exzess)* addiert wird und statt des Exponenten die Summe gespeichert wird.
  - Diese Summe ist dann eine vorzeichenfreie positive Zahl.
- Vorteil der Biased-Darstellung, dass so ein Größenvergleich zwischen zwei positiven Gleitkommazahlen erleichtert wird.
  - Es genügt, die Ziffernfolgen *em*, also jeweils Exponent *e* gefolgt von Mantisse *m*, miteinander zu vergleichen.
  - Eine Gleitkomma-Subtraktion mit anschließendem Vergleich auf Null wäre weitaus aufwändiger.
  - Nachteil ist, dass nach Addition zweier Biased-Exponenten der Bias wieder subtrahiert werden muss, um das richtige Ergebnis zu erhalten.



Typ	Größe (1+r+p)	Exponent (r)	Mantisse (p)	Werte des Exponenten (e)	Biaswert (B)
single	32 bit	8 bit	23 bit	$-126 \leq e \leq 127$	127
double	64 bit	11 bit	52 bit	$-1022 \leq e \leq 1023$	1023

Beispiel

Umwandlung von $1234567{,}89_{10}$ ins IEEE -754 Format

$1234567{,}89_{10} = 2^{20{,}23557474}$

$= 2^{0{,}23557474} * 2^{20}$

$= 1{,}177375686_{10} * 2^{20}$

$= 1{,}00101101011010000111111000110010_2 * 2^{20}$

= 20 mal geteilt durch 2

Bias

1 bit	=> V = 0
8 bit	Exponent = 0001 0100_2
23 bit	Mantisse = ,001 0110 1011 0100 0011 1111_2
32 bit	

23

Beispiel

Umwandlung von $1234567{,}89_{10}$ ins IEEE -754 Format

$$1234567{,}89_{10} = 2^{20{,}23557474}$$
$$= 2^{0{,}23557474} * 2^{20}$$
$$= 1{,}177375686_{10} * 2^{20}$$
$$= 1{,}00101101011010000111111000110010_2 * 2^{20}$$

Nächste Folie

1 bit	=> V = 0
8 bit	Exponent = 0001 0100$_2$ + 0111 1111$_2$ = 1000 1011
23 bit	Mantisse = ,001 0110 1011 0100 0011 1111$_2$
32 bit	em = 1000 1011 001 0110 1011 0100 0011 1111$_2$

Bias

vergleichbar

Der Exponent wird als nichtnegative Binärzahl gespeichert, indem man den festen Biaswert B addiert.

Der Biaswert dient dazu, dass negative Exponenten durch eine vorzeichenlose Zahl (die Charakteristik) gespeichert werden können, unter Verzicht auf alternative Kodierungen wie z. B. das Zweierkomplement.

Beispiel zur Mantissenberechnung

- Die 1. Stelle nach dem Komma repräsentiert die Vielfachen von $2^{-1} = 1/2$, die 2. Stelle die Vielfachen von $2^{-2} = 1/2^2$ usw.
- Beispiel: ,1011

$$\frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} = \frac{8 \cdot 1 + 0 + 2 \cdot 1 + 1}{16} = \frac{8 + 2 + 1}{16} = \frac{11}{16} = 0{,}6875_{(10)}$$

- Die Rück-Rechnung in einen binären Nachkommawert erfolgt, indem man die Nachkommazahl mit 2 multipliziert und dann prüft, ob sie größer 1 ist
  - → dann ist sie größer als 0,5 und die erste binäre Nachkommastelle eine 1
  - → den Rest nimmt man wieder mal 2, um zu prüfen, ob er größer 1 ist usw.

  - $2 \cdot 0{,}6875 = 1{,}375$ → Ziffer: 1
  - $2 \cdot 0{,}375 = 0{,}75$ → Ziffer: 0
  - $2 \cdot 0{,}75 = 1{,}5$ → Ziffer: 1
  - $2 \cdot 0{,}5 = 1$ → Ziffer: 1
  - → Resultat: ,1011

Quelle: http://www.arndt-bruenner.de/mathe/scripts/Zahlensysteme.htm

Mantissenberechnung

Zahlensystem	Ziffernfolge
2 (binär)	0,00101101011010000111111000110010011110011011111010100 0...
10 (dezimal)	,177375686

Klicke für eine Erläuterung des Rechenweges der letzten Umwandlung auf diesen Button:

Wie geht das? ☐ bei jeder Eingabe erklären

```
Der Dezimalbruch  0,177375686  soll ins 2er System umgewandelt werden.

Gehe nach folgendem Verfahren vor, um die Nachkommaziffern zu erhalten:
 (1) Multipliziere die Zahl mit der Basis 2
 (2) Die Zahl vor dem Komma ist die nächste Ziffer des Ergebnisses
 (3) Schneide die Zahl vor dem Komma weg.
 (4) Wiederhole ab (1), bis der Rest 0 ist, sich ein Rest wiederholt
     oder die gewünschte Genauigkeit erreicht ist.

    2 · 0,177375686 = 0,354751372    --> Ziffer: 0
    2 · 0,354751372 = 0,709502744    --> Ziffer: 0
    2 · 0,709502744 = 1,419005488    --> Ziffer: 1
    2 · 0,419005488 = 0,838010976    --> Ziffer: 0
    2 · 0,838010976 = 1,676021952    --> Ziffer: 1
    2 · 0,676021952 = 1,352043904    --> Ziffer: 1
    2 · 0,352043904 = 0,704087808    --> Ziffer: 0
    2 · 0,704087808 = 1,408175616    --> Ziffer: 1
    2 · 0,408175616 = 0,816351232    --> Ziffer: 0
    2 · 0,816351232 = 1,632702464    --> Ziffer: 1
    2 · 0,632702464 = 1,265404928    --> Ziffer: 1
    2 · 0,265404928 = 0,530809856    --> Ziffer: 0
```

$$= 1{,}177375686_{10} * 2^{20}$$
$$= 1{,}00101101011010000$$

Ariane 5: Jungfernflug

Wie das Unglück passierte …

- *On 4th June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher … exploded.*
- *A 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.*
- *The software ended up triggering a system diagnostic that dumped its debugging data into an area of memory being used by the programs guiding the rocket's motors. At the same time, control was switched to a backup computer that unfortunately had the same data. This was misinterpreted as necessitating strong corrective action and the rocket's motors swiveled to the limits of their mountings. Disaster ensued.*

Pentium FDIV-Bug

FDIV-Bug bezeichnet einen Hardwarefehler des Pentium-Prozessors von Intel. Der Fehler wurde im November 1994 anderthalb Jahre nach der Markteinführung bekannt und führt bei Gleitkomma-Divisionen mit bestimmten, relativ wenigen Wertepaaren zu ungenauen Ergebnissen.[1] Kein anderer Fehler in einem CPU-Design hat so viel Wirbel und Aufregung bei Anwendern und Fachleuten ausgelöst. In der Folge werden entdeckte Hardwarefehler von vielen Herstellern veröffentlicht. Vielen Anwendern ist dadurch bewusst geworden, dass komplexe Hardware, ebenso wie Software, typischerweise zahlreiche Fehler hat.

Die Bezeichnung *FDIV-Bug* leitet sich vom Namen eines häufig verwendeten Gleitkommabefehls bei x86-Prozessoren ab.

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.3338204491362410\,02$$

The world of mathematics.

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.3337390689020375\,89$$

The world from the Pentium's point of view.

On **December 19.9999973251**, 1994, Intel announced a total free replacement of all faulty Pentium processors on the basis of request. Andrew Grove made public apologies. The whole affair cost Intel $475,000,000, which was more than half of their profit made within the last quarter of 1994.

Schachfeld

Schachfeld → Feld von 8 mal 8

```
Schachfeld :=
(
      („Turm_S“ ,„Springer_S“,„Läufer_S“, … ,„Turm_S“),
      („Bauer_S“,„Bauer_S“ ,„Bauer_S“ , … ,„Bauer_S“),
      („Leer“ ,„Leer“, ,„Leer“ , … ,„Leer“),
      („Leer“ ,„Leer“, ,„Leer“ , … ,„Leer“),
      („Leer“ ,„Leer“, ,„Leer“ , … ,„Leer“),
      („Leer“ ,„Leer“, ,„Leer“ , … ,„Leer“),
      („Bauer_W“,„Bauer_W“ ,„Bauer_W“ , … ,„Bauer_W“),
      („Turm_W“ ,„Springer_W“,„Läufer_W“, … ,„Turm_W“)
)
```

Schachfeld := array(8,8) of String

Arrays

- Array (Feld): Mehrere Werte eines Datentyps in eine Variable ablegen.
  - Array Variable belegt zusammenhängenden Speicherbereich
  - Mehrdimensionale Felder z.B. 3-dimensionales für Volumen
  - Vektor: 1-dimensionales Array

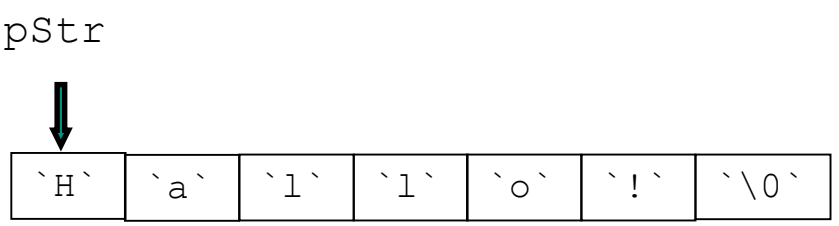
```
char str[7] = { `H`, `a`, `l`, `l`, `o`, `!`, `\0`};
```

- oder äquivalent

```
char str[7] = "Hallo!";
```

- Array Variable ist auch Zeiger (Pointer) auf das erste Element des Arrays im Speicher

```
char* pStr = str;
```



Arrays, weitere Beispiele

- Ein Array vollständig mit definierten Werten initialisieren

```
int iArr[5] = {1, 2, 3, 4, 5};
```

- Sub-Menge eines Array mit definierten Werten initialisieren, den Rest des Arrays mit default Wert (=0):

```
double dArr[4] = {1.1, 2.2};
```

- Alle Elemente eines Array mit default Wert initialisieren

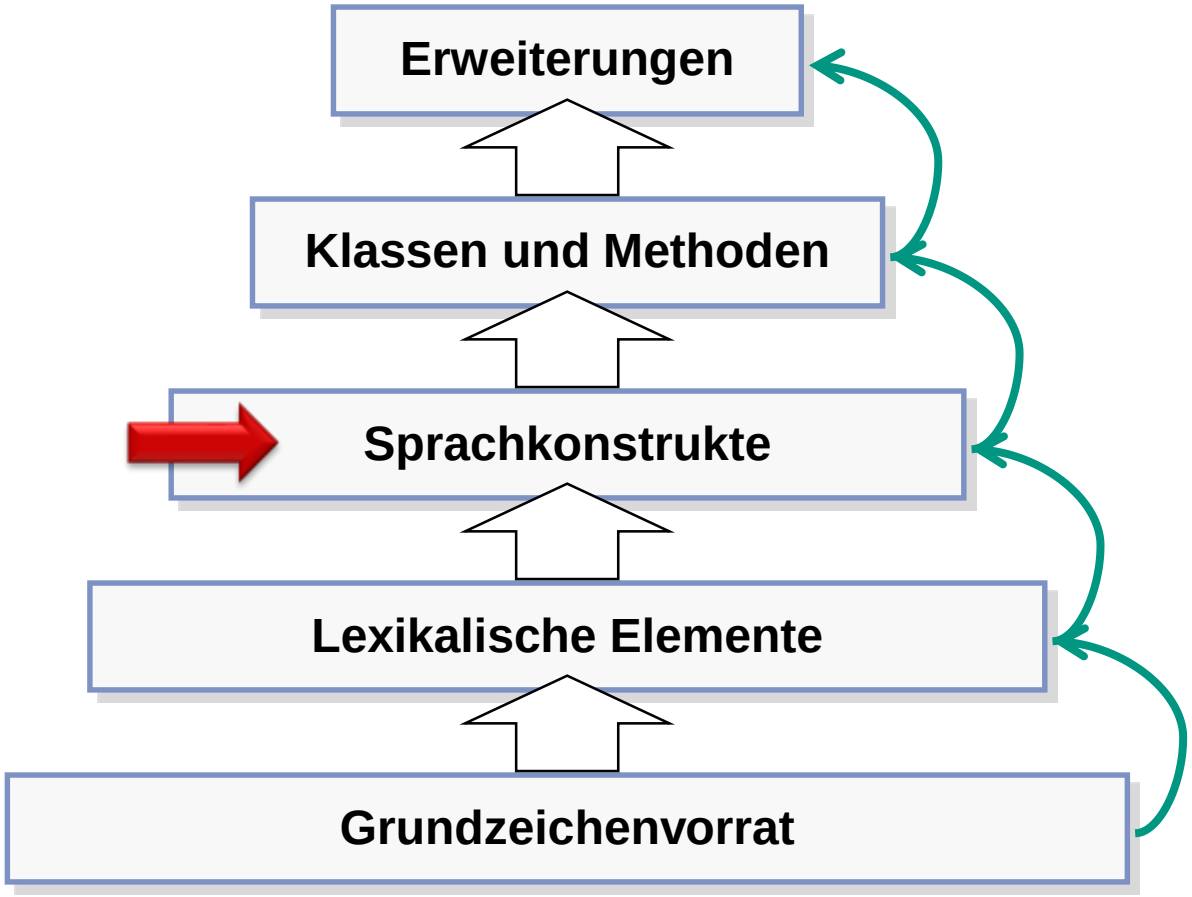
```
long lArr[100] = {};
```

- Zugriff auf Array Elemente über Index-Operator

```
int temp = iArr[2];      // temp = 3
iArr[2] = iArr[3];       // iArr[2] = 4
iArr[3] = temp;          // iArr[3] = 3
```

Kern von C++ / Sprachaufbau

Kern von C++
(Vordefinierte Datentypen, Operatoren, Kontrollstrukturen)



C++ Operatoren

Kern von C++
(Vordefinierte
Datentypen,
Operatoren,
Kontrollstrukturen)

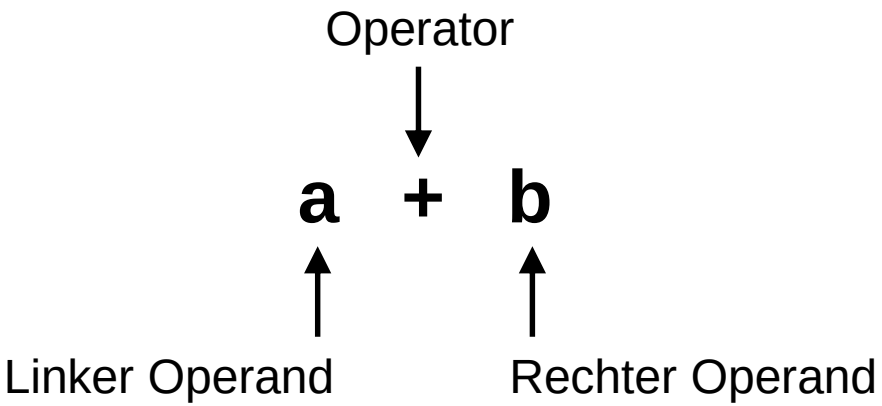
binäre und unäre arithmetische
Operatoren

Vergleichsoperatoren

Logische Operatoren

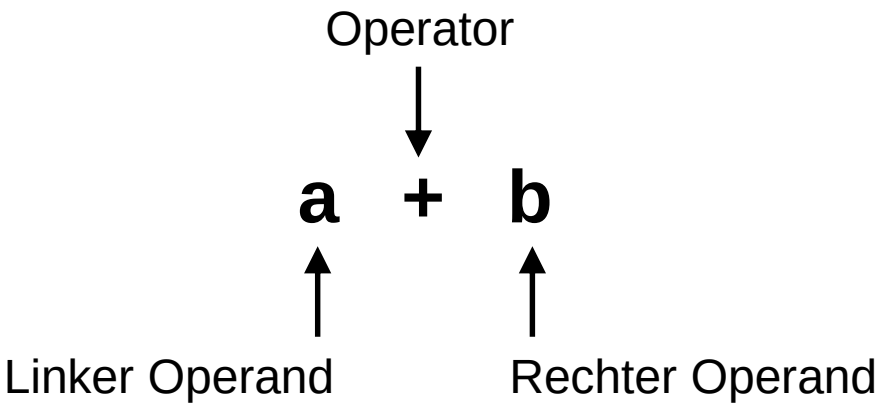
Logische Bitoperatoren

Operatoren und Operanden



Binärer Operator und Operanden

Bei binären Operatoren treten zwei Operanden auf.



Die binären arithmetischen Operatoren

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulodivision

Arithmetische Operatoren

x+y	// plus	***18***
+x	// Vorzeichen plus	***+12***
x−y	// minus	***6***
−x	// Vorzeichen minus	***-12***
x∗y	// multiply	***72***
x/y	// divide	***2***

x+=y	// x = x+y	***18***
++x	// inkrement: x = x+1	***13***
x−=y	// x = x-y	***6***
−−x	// dekrement: x = x-1	***11***
x∗=y	// x = x*y	***72***
x/=y	// x = x/y	***2***
x%=y	// x = x%y (modulo)	***2 + 0***

Beispiel: x = 12, y = 6

Operatoren

x+y // plus

+x // Vorzeichen plus

x−y // minus

−x // Vorzeichen minus

x∗y // multiply

x/y // divide

x+=y // x = x+y

++x // inkrement: x = x+1

x−=y // x = x-y

−−x // dekrement: x = x-1

x∗=y // x = x*y

x/=y // x = x/y

x%=y // x = x%y (modulo)

Priorität	Operator
hoch ↕ niedrig	`++ --`
	`+ -` (Vorzeichen)
	`* / %`
	`+` (Addition)
	`-` (Subtraktion)

Vorrang der arithmetischen Operatoren

Vergleichsoperatoren

Operator	Bedeutung
`<`	kleiner
`<=`	kleiner gleich
`>`	größer
`>=`	größer gleich
`==`	gleich
`!=`	ungleich

Beispiel:

Vergleich	Ergebnis
`5>=6`	False
`1.7<1.8`	True
`4+2==5`	False
`2*4!=7`	True

Vorrang der Vergleichsoperatoren

Priorität	Operator
hoch ↕ niedrig	arithmetische Operatoren
	`<` `<=` `>` `>=`
	`==` `!=`
	Zuweisungsoperatoren

Logische Operatoren

Operator	Bedeutung
`&&`	UND
`||`	ODER
`!`	NICHT

Boolesche Operatoren für zusammengesetzte Bedingungen

Beispiel:

x	y	logischer Ausdruck	Ergebnis
1	-1	`x <= y || y >= 0`	false
0	0	`x > -2 && y == 0`	true
-1	0	`x && !y`	true
0	1	`!(x+1) || y-1 > 0`	false

s. unten

Wahrheitstafel

A	B	!A	A && B	A || B
`true`	`true`	false	true	true
`true`	`false`	false	false	true
`false`	`true`	true	false	true
`false`	`false`	true	false	false

Die Operanden der booleschen Operatoren sind vom Typ bool. Als Operanden sind aber auch beliebige Ausdrücke zulässig, deren Typ in bool konvertiert werden kann (alle arithmetischenTypen). Operand wird in false konvertiert, wenn sein Wert 0 ist, jeder von 0 verschiedene Wert wird zu true konvertiert!

Logische Bitoperatoren

- Für bitcodierte Daten einzelne Bits lesen oder ändern oder wortweise verarbeiten, daher hohe Speicherplatzeffizienz und Rechenzeiteffizienz
- Operanden müssen ganzzahligen Datentyp haben
- Nicht verwechseln mit boolschen (logischen) Operatoren
- Bei Linksschieben
  - „nachschieben“ mit 0-Bits
- Bei Rechtsschieben
  - Für unsignierte Zahlen werden die durch den Verschiebevorgang frei gewordenen Bitpositionen mit 0 angefüllt.
  - Für signierte Zahlen wird das Vorzeichenbit verwendet, um die frei gewordenen Bitpositionen zu füllen. In anderen Worten, wenn die Zahl positiv ist, wird 0 verwendet, und wenn die Zahl negativ ist, wird 1 verwendet.

Operator	Bedeutung
&	UND
|	ODER
~	NICHT
^	Exklusiv ODER
<<	Links-Shift
>>	Rechts-Shift

Unsigned integer a, b, c;	Bitmuster
a = 5;	0 0 . . . . . 0 0 0 0 0 1 0 1
b = 12;	0 0 . . . . . 0 0 0 0 1 1 0 0
c = ~a;	1 1 . . . . . 1 1 1 1 1 0 1 0
c = a & b;	0 0 . . . . . 0 0 0 0 0 1 0 0
c = a | b;	0 0 . . . . . 0 0 0 0 1 1 0 1
c = a ^ b;	0 0 . . . . . 0 0 0 0 1 0 0 1
c = b << 3;	0 0 . . . . . 0 1 1 0 0 0 0 0
c = b >> 2;	0 0 . . . . . 0 0 0 0 0 0 1 1

Beispiel: bitweise AND

```
/* and.c */
#include <stdio.h>
int main(void) {
        int x=55;
        cout << x << endl;
        // x=55
        x= x&7;
        cout << x;
        // x=7
        return 0;
}
```

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	128	64	32	16	8	4	2	1
55	0	0	1	1	0	1	1	1
&7	0	0	0	0	0	1	1	1
7	0	0	0	0	0	1	1	1

Beispiel: bitweise AND

- Mit dem bitweisen UND-Operator lässt sich testen, ob eine Zahl gerade oder ungerade ist.
- Es muss nur Bit 0 (bzw. das LSB) daraufhin überprüft werden, ob es gesetzt (ungerade, also = 1) oder nicht gesetzt (gerade, also = 0) ist.
- Folgendes Beispiel demonstriert dies:

```
/* gerade.c */
#include <stdio.h>
int main(void) {
        int x;
        cout << "Bitte geben Sie eine Zahl ein:“ << endl;
        cin >> x;
        if(x&1) // Ist das erste Bit gesetzt?
                cout << "ungerade“<< endl;
        else // Nein, es ist nicht gesetzt.
                cout << “gerade"<< endl;
        return 0;
}
```

Beispiel: bitweise AND

- Mit dem bitweisen UND-Operator lässt sich testen, ob eine Zahl gerade oder ungerade ist.
- Es muss nur Bit 0 (bzw. das LSB) daraufhin überprüft werden, ob es gesetzt (ungerade, also = 1) oder nicht gesetzt (gerade, also = 0) ist.
- Beispiel:

55	0	0	1	1	0	1	1	1
&1	0	0	0	0	0	0	0	1

```
/* gerade.c */
#include <stdio.h>
int main(void) {
        int x;
        cout << "Bitte geben Sie eine Zahl ein:" << endl;
        cin >> x;
        if(x&1) // Ist das erste Bit gesetzt?
                cout << "ungerade"<< endl;
        else // Nein, es ist nicht gesetzt.
                cout << "gerade"<< endl;
        return 0;
}
```

Beispiel: bitweise AND

- Mit dem bitweisen UND-Operator lässt sich testen, ob eine Zahl gerade oder ungerade ist.
- Es muss nur Bit 0 (bzw. das LSB) daraufhin überprüft werden, ob es gesetzt (ungerade, also = 1) oder nicht gesetzt (gerade, also = 0) ist.
- Beispiel:

55	0	0	1	1	0	1	1	1
&1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1

```
/* gerade.c */
#include <stdio.h>
int main(void) {
        int x;
        cout << "Bitte geben Sie eine Zahl ein:“ << endl;
        cin >> x;
        if(x&1) // Ist das erste Bit gesetzt?
                cout << "ungerade“<< endl;
        else // Nein, es ist nicht gesetzt.
                cout << “gerade"<< endl;
        return 0;
}
```

C++ Operatoren

Kern von C++
(Vordefinierte
Datentypen,
Operatoren,
Kontrollstrukturen)

binäre und unäre arithmetische

Anweisungen zur Kontrolle des Programmflusses

Schleifen mit while, do-while und for

Verzweigung mit if-else und switch

Bedingungsfreie Sprünge mit continue, break

Verzweigung, Einfach-Entscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird

- Syntax:

```
if( Bedingung ) {
  AnweisungA;
} else {
  AnweisungB;
}
```

Optional

- **`else`**-Verzweigung ist optional
- Beispiel (*Differenz als Betrag*):

```
if( a < b ) {
  c = b - a;
} else {
  c = a - b;
}
```

Verzweigung, Mehrfach-Entscheidung

- Festlegen, welche Anweisung als nächstes ausgeführt wird
  - `switch case` anstelle geschachtelter `if else`
- Syntax:

```
switch( Ausdruck ) {
        case 1:
        AnweisungA;
        break;

        case 2:
        AnweisungB;
        break;
      …
        default:
        AnweisungC;
}
```

Optional

Beispiel:

```
char alpha;
...
switch (alpha)
{
case 'e':
case 'E':
   Aktion1;
break;
case 'a':
case 'A':
   Aktion2;
}
```

  - „Ausdruck“ muss vom Typ ganzzahlig oder char sein!

While - Schleife

- Viele Anweisungen müssen mehrfach wiederholt werden
  - Feste Anzahl Wiederholungen oder abhängig von einer Bedingung
- **`while`**-Schleife - Kopfgesteuert
  - Bedingung **`true`** oder **`false`**
- Syntax:

```
while( Bedingung ) {
          Anweisung
       }
```

- Beispiel:

```
int zahl = -1;
       while( zahl <= 0 ) {
          cout << "Bitte geben Sie eine";
          cout << "Zahl größer Null ein: ";
          cin >> zahl;
       }
```

Do-While - Schleife

- Auch Fußgesteuerte Schleife genannt
  - Wird mindestens einmal ausgeführt (Unterschied zur **`while`**-Schleife)

- Syntax:
```
do {
    Anweisung
} while( Bedingung );
```

- Beispiel:
```
int zahl = 1;
    do {
      cout << "Bitte geben Sie eine";
      cout << "Zahl größer Null ein: ";
      cin >> zahl;
    } while( zahl <= 0 );
```

For – Schleife(1)

- Oft verwendet bei fester Anzahl an Durchläufen oder wenn eine Zählvariable benötigt wird
- Syntax:

```
for( Ausdruck1; Ausdruck2; Ausdruck3 ) {
        Anweisung;
}
```

- Ausdruck1 = Initialisierung
  - Zu Beginn der Schleife einmalig ausgeführt
- Ausdruck2 = Bedingung
  - Schleife läuft, solange Bedingung wahr
- Ausdruck3 = Veränderung
  - Am Ende jedes Schleifen-Durchlaufs ausgeführt

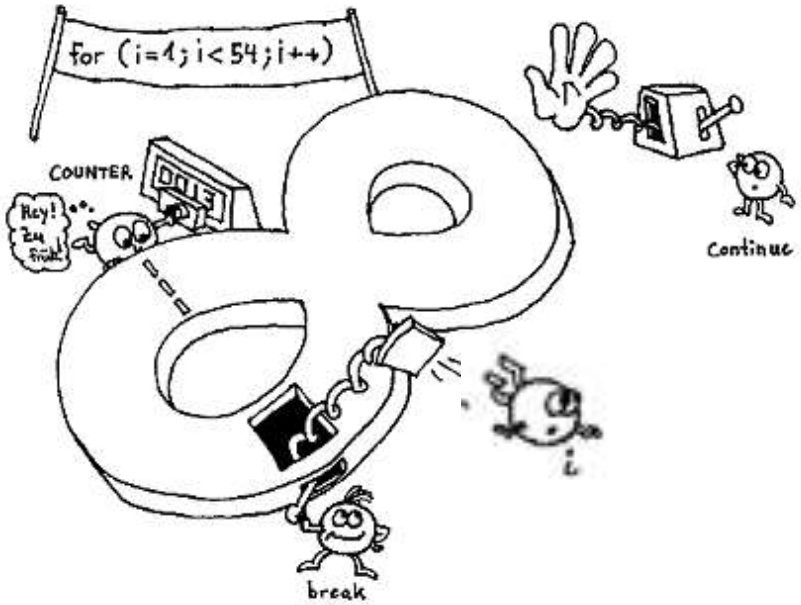
```
For ( int i = 1; i < 5; i++ )
{
        cout << i << " " << i * i;
        cout << endl;
}
```

```
1   1
2   4
3   9
4  16
```

For – Schleife(2) und break;

- Um eine Schleife sofort zu verlassen kann der Befehl break verwendet werden
  - Nützlich für besondere Ereignisse und Fehler-Behandlung
- Beispiel:

```
for( int i = 1; i < 50; i++ ) {
        cout << i << " " << i * i;
        cout << endl;
        if( i * i > 1000 ) {
          break;
        }
}
```




```
... ...
30  900
31  961
32  1024
break
```

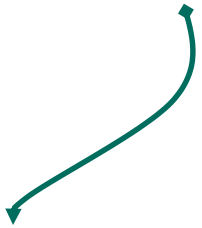
For – Schleife(2) und break;

- Beispiel:

```
int main ()
{
   char inp;
   ...
   for (;;)     // Endlos-Schleife
   {
      ...
      // Ein Auswahlmenü ausgeben
      cout << "Auswahl? ";
      // Antwort einlesen
      cin >> inp;
      if (inp == 'E')       // Falls Auswahl E
          break;            // Schleife verlassen
      ...
   }
   ...
}
```

Schleifen und continue;

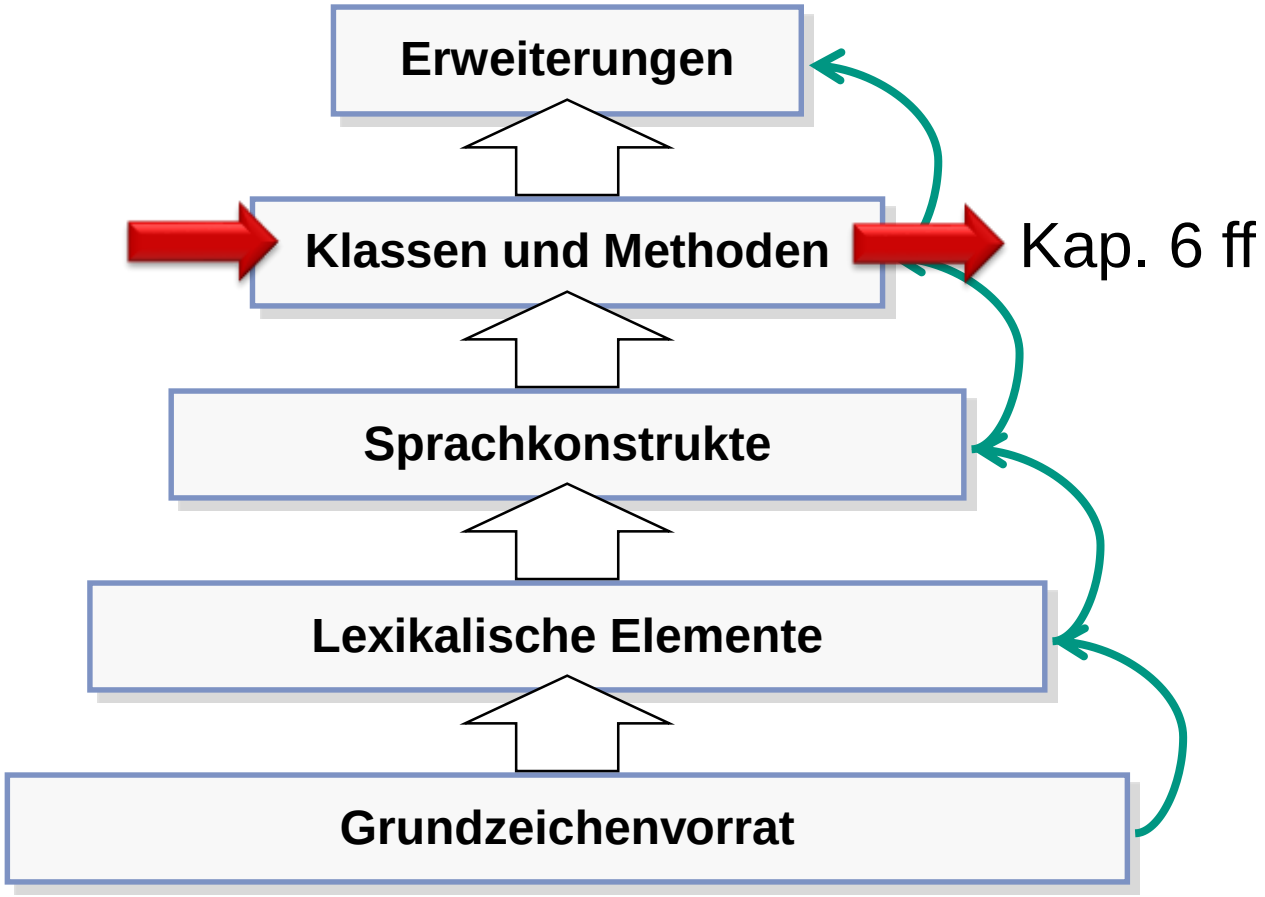
- Die continue-Anweisung ist nur innerhalb einer for- oder while-Schleife erlaubt.
- Sie bewirkt, dass die restlichen, der continue-Anweisung folgenden Anweisungen, übersprungen werden
  - Die Schleife selbst wird aber nicht verlassen.
- Bei einer for-Schleife wird nach der continue-Anweisung mit der Auswertung des letzten Ausdrucks in der for-Klammer (Aktion pro Schleifendurchlauf) fortgefahren.
- Beispiel:



```
int main ()
{
    ...
    for (int index=0; index<10; index++)
    {
        ...
        if (index == 5)  // Falls index gleich 5
            continue;    // Rest der Schleife überspringen
        ...              // Weitere Anweisungen wenn
                         // index ungleich 5
        ...
    }
    ...
}
```

Kern von C++ / Sprachaufbau

Kern von C++
(Vordefinierte Datentypen, Operatoren, Kontrollstrukturen)



- Programmablauf Verzweigung
  - if und switch case
- Programmablauf Schleifen
  - While-Schleife
  - Do-While-Schleife
  - For-Schleife
  - break, continue


Backup
Shift

Betriebssystem

- Betriebssystem als Schnittstelle für Systemfunktionen (Systemaufrufe), die von Programmen genutzt werden kann.
  - Hierzu sind auch Boot-Loader, Gerätetreiber, bestimmte Systemdienste, Programmbibliotheken etc. erforderlich.
- Aufgaben eines Betriebssystems
  - Anpassung der Leistungen der Hardware an die Bedürfnisse der Benutzer (Abstraktion der Hardware):
    - Das Betriebssystem erweitert die Hardware Funktionalität, z.B. durch Dateiverwaltung.
  - Organisation, Steuerung und Kontrolle des gesamten Betriebsablaufs im System:
    - Zuordnung der Benutzeraufträge zu entsprechenden Ausführungseinheiten (z.B. Prozesse)
    - geeignete Ablaufplanung unter Beachtung möglicher Wechselwirkungen zwischen den Ausführungseinheiten.
  - Verwaltung und ggf. geeignete Zuteilung von Betriebsmitteln (Ressourcen) an verschiedene Ausführungseinheiten.
  - Kontrolle und Durchsetzung von Schutzmaßnahmen (z. B. Zugriffsrechte), insbesondere zur Sicherung der Integrität von Daten und Programmen, vor allem im Mehrnutzer-Betrieb.
  - Protokollierung des gesamten Ablaufgeschehens im System

Betriebssystem: Klassifikation

- nach Betriebsart
  - Stapelverarbeitung (batch processing)
    - z.B. IBM OS/390, z/OS, BS2000
  - Dialogbetrieb (interactive processing)
    - z.B. MS-DOS, MS-Windows, UNIX, LINUX
  - Echtzeitverarbeitung (real time processing)
    - z.B. VxWorks, VRTX, QNX, OSEK/VDX
  - Verteilte Verarbeitung (distributed processi
    - z.B. Amoeba, MACH, Novell Netware
- nach Anzahl der Benutzer
  - Einzelnutzer-System (Single User System)
    - z.B. MS-DOS, MS Windows 95/98/ME
  - Mehrnutzer-System (Multi User System)
    - z.B. UNIX, Linux, IBM z/OS, BS 2000, OpenVMS
- nach Anzahl der Aufträge
  - Einzelprozess-System (Single Tasking System)
    - z.B. CPM, MS-DOS
  - Mehrprozess-System (Multi Tasking System)
    - z.B. UNIX, Linux, IBM z/OS, BS 2000, OpenVMS, VxWorks, VRTX, LynxOS

(Desktop)