

Programm-Struktur

**Programm-Struktur**

```
/* Präprozessor-Direktiven */  
/* globale Deklarationen: Typen, Klassen, Variablen */  
/* Funktionsdeklarationen oder -definitionen */  
  
int main()  
{  
    /* Deklaration lokaler Variablen */  
    /* Anweisungen */  
}  
/* Funktionsdefinitionen (falls oben deklariert) */
```

**Modulare Programmierung (Vorschlag!)**

→ Struktur der Quellcode-Datei dateiname.cpp

```
#include "dateiname.h"  
/* Funktionsdefinitionen (= ~implementierungen) */
```

→ Struktur der zugehörigen Header-Datei dateiname.h

```
#ifndef DATEINAME_H // Wächter gegen  
#define DATEINAME_H_ // mehrfache Deklaration  
/* (weitere) Präprozessor-Direktiven */  
/* Definition von Typen, Strukturen, ... */  
/* Funktionsprototypen (= ~deklarationen) */  
#endif // Ende des Wächters DATEINAME_H
```

**ISO-C++: Header für Standard-Bibliotheken**

Die Funktionalitäten sind im Namensraum std deklariert.

algorithm array bitset complex deque exception fstream  
functional iomanip ios iosfwd istream iterator  
limits list locale map memory new numeric ostream queue  
random regex set sstream stack stdexcept streambuf string

**Zeichenkonstanten** ( $\text{o}=\text{Oktalziffer}$ ,  $\text{h}=\text{Hexadezimalziffer}$ ,  $\text{TP}=\text{Typ-Präfix}$ )

Zeichenkonstanten						
(o=Oktalziffer, h=Hexadezimalziffer, TP=Typ-Präfix)						
direkt	TP' a'	Oktal	TP' \ooo'	hexadezimal	TP' \xhh'	
				1	char	
TP:	(ohne)					
		Anzahl der Zeichen		≥2	int	
	L	zwischen den :		≥1	wchar_t	
	u			1	char16_t	
	U			1	char32_t	
Sonderbehandlung:	'	'	"	'\n'	?	'\?
universelle Zeichennamen (Unicode)				\uhhhh		\Unhhhhhh
Nicht für Basis-Zeichensatz verwenden!						
Beispiel: #pragma listing on "\\"\\listing dir\\\""	bei char: EOF	bei wchar_t: WEOF				
Pragma-Operator	Pragma (Direkt String)	Dateiende-Zeichen				
Beispiel (Wirkung wie oben): Pragma [listing on "\\"\\listing dir\\\""]						

## ASCII-Zeichensatz (Hex-Codes 00-7F) (SpalteZeile)

Dynamical

C++-Präprozessor

**Bezeichner** (Identifier)

- 1. Zeichen: lateinisches Alphabet, Unterstrich \_
- weitere Zeichen: auch Dezimalziffern u. universelle Zeichennamen
- Groß- und Kleinbuchstaben werden unterschieden!
- keine Schlüsselwörter
- Makro-/Typ-/Klassen-/Funktionsnamen der Standardbibl. vermeiden

**Geltungsbereiche** (Teil d. Programms, in dem der Bez. sichtbar ist):

- **lokal**: innerhalb eines Blocks deklarierte Bez. (keine Marken),
- **Funktionsprototyp**: Parameter gültig bis Ende des Prototyps,
- **Funktion**: Bezeichner von (Sprung-)Marken,
- **Namensraum**: → siehe „Namensräume“,
- **Klasse**: innerhalb einer Klasse, wenn kein lokaler Geltungsbereich,
- **Aufzählung**: innerhalb einer „scoped enumeration“,
- **Datentyp**: Bez. außerhalb von Blöcken, Klassen oder Namensräumen.

**Schlüsselwörter (reservierte Bezeichner)**

```
alignas alignof and and_eq asm auto bitand bitor bool break case
catch char char16_t char32_t class compl const constexpr
const_cast continue decltype default delete do double
dynamic_cast else enum explicit export extern false float for
friend goto if inline int long mutable namespace new not not_eq
nullptr operator or or_eq private protected register
reinterpret_cast return short signed sizeof static
static_assert static_cast struct switch template this throw
true try typeid typeid typename union unsigned using virtual
void volatile wchar_t while xor xor_eq
```

**Namensräume (Namespaces)**

- Ein Namensraum definiert einen benannten Sichtbarkeitsbereich.
- in keinem Nam.raum deklarierte Namen → **globaler Namensraum**
- C++-Standardbibliothek: Namensraum **std**.
- unbenannter Namensraum: Namen haben Datei-Sichtbarkeit

```
namespace [nsNamespace]
```

**/ Variablendeklarationen, Funktionsprototypen, Namensräume**

```
} namespace eins { int var = 5; int x = 7; }
namespace zwei { double var = 1.4142; }

Beispiel 1: ...
cout << eins::var; // → Ausgabe: 5
cout << zwei::var; // → Ausgabe: 1.4142

namespace aussen { int x = 10;
    namespace innen { double x = 3.14; } }
```

**Beispiel 2:**

```
cout << aussen::x; // → Ausgabe: 10
cout << aussen::innen::x; // → Ausgabe: 3.14
```

**using-Deklaration**

- Fügt einen **Namen** zu dem Sichtbarkeitsbereich hinzu, in dem sich die using-Deklaration befindet. → Gestattet Verwendung eines Namens aus einem Namensraum ohne explizite N.r.-Qualifikation.

```
using nsNamespace::name;
```

```
Beispiel 1: ...
using zwei::var;
cout << var; // → Ausgabe: 1.4142
```

**using-Anweisung**

- Gestattet Verwendung **aller Namen** aus einem Namensraum, ohne daß dieser Namensraum qualifiziert werden muß.

```
using namespace nsNamespace;
```

```
Beispiel 1: ...
using namespace eins;
cout << var; // → Ausgabe: 5
```

**Operatoren** (geordnet nach fallender Priorität P)

- **Z**=Zusammenfassung (Assoziativität): L=von links, R=von rechts, K=keine „Beschreibung“ beschreibt das Verhalten bei intrinsischen C++-Typen.
- **Ü**=Überladbarkeit: ja, n/nein

P	Beschreibung	Operator $\otimes$	Ü	Z
1	Sichtbarkeits-(Scope)- Bereichsauflösungsooperator	$::$	n	K
	Klammern (Wertkonstruktion)	$(x+y)*z$	n	
	Funktionsaufruf	$fktname (param)$	j	
	Typumwandlung	$\text{einfaller\_typ} (\text{ausdruck})$	n	
	Subskript-Operator (Vektor-Element)	$feld[\text{index}]$	j	
2	Pfeiloperator (struct/union/class-Zeiger)	$zeig->\text{mem}$	j	
	Punktoperator (struct/union/class-Member)	$\text{name}.mem$	n	L
	Post-In-/Dekrementierung	$x++$	x--	
	Typinformation	$\text{typred} (\text{ausdruck od. typ})$	j	
	Typumwandlung	$\text{dynamic\_cast static\_cast}$	n	
	Prä-In-/Dekrementierung	$++x$	--x	
	Vorzeichen: Plus, Minus	$+x$	-x	
	Negation: logisch, bitweise	$\text{!}x$	$\sim x$	
	Verweisoperator (Indirektion)	$*\text{zeiger}$		
	Adreß-Operator	$\&\text{name}$		
	Größe eines Objekts	$\text{std::size\_t sizeof objekt}$		R
	Datentyp-Größe	$\text{std::size\_t sizeof (typ)}$		
	Speicher-Allokation	$\text{new typ} [\text{n}]$		
	Speicher-Allokation	$\text{delete} [\text{l}] \text{ zeiger}$	j	
4	Typumwandlung (C-Cast)	$(typ) ausdruck$		
	Zeiger-auf-Member-Auswahl	$\text{z name}*\text{z member}$		
	Multiplikation, Division, Modulo-Div.	$\text{x}*y$	$\text{x}/y$	
	Addition, Subtraktion	$\text{x}+\text{y}$	$\text{x}-\text{y}$	
	bitweises Links-, Rechts-Schieben um n Bit	$\text{m}<\text{n}$	$\text{m}>\text{n}$	
	relationale Operatoren	$\text{x}>\text{y}$	$\text{x}<\text{y}$	
	Gleichheitsooperatoren	$\text{x}==\text{y}$	$\text{x}!=\text{y}$	
	bitweises UND (AND)	$\text{m&n}$		
	bitweises Exklusiv-ODER (XOR)	$\text{m}^{\wedge}\text{n}$		
	bitweises Inklusiv-ODER (OR)	$\text{m} \text{n}$		
	logisches UND	$\text{m&n}$		
	logisches ODER	$\text{m} \text{n}$		
	bedingter Ausdruck	$\text{test ? wert\_bei\_wahr : sonst}$		
	Zuweisungs-	$\text{x=y}$	$\text{x}=\text{y}$	
	operatoren	$\text{m}=<\text{n}$	$\text{m}=>\text{n}$	
		$\text{m}=\text{n}$	$\text{m}!=\text{n}$	
	Sequenzoperator (erst x, dann y auswerten)	$\text{x},\text{y}$		
	konstanter Ausdruck			
				K
19	konstanter Ausdruck			R
				J
				L

**Operator  $\otimes$  als Funktionsaufruf: Nichtelementfunktionen**

- Beispiel: Aufruf **s=operator+(x,y)** ist identisch zu **s+x.y**.

```
Syntax           Ersetzung durch          Syntaxis
x*y             x. operator<math>(\mathbf{y})</math>      x (arg)
@x              x. operator<math>(@)</math>        x [arg]
@x              x. operator<math>[@]</math>       x [idx]
@x              x. operator<math>[]</math>        x (idx)
@x              x. operator<math>->(&lt;&gt;)</math>     x. operator<math>->(&lt;&gt;)
@x              x. operator<math>=(&lt;&gt;)</math>      (Typ) x x. operator<math>=(&lt;&gt;)
```

Nur zur studiumsbegleitenden Verwendung durch Professoren, Mitarbeiter und Studenten der HTW Dresden freigegeben. Keine Gewähr auf Vollständigkeit oder Korrektheit. Keine Haftung. Hinweise/Kritiken willkommen.

**Überladen von Operatoren**

prinzipiell wie bei „Funktionen: Überladen“	• Definition wie bei „Funktionen: Überladen“
keine Default-Argumente erlaubt (Ausnahme Fktсаufruft-Op. ():)	• Default-Argumente möglich, kann beliebig viele Parameter haben.)
überladbare Operatoren: siehe Tabelle (Spalte <b>Ü</b> )	• überladbare Operatoren: siehe Tabelle (Spalte <b>Ü</b> )
Default-Argumente möglich, Priorität, Assoziativität, Parameteranzahl: nicht veränderbar	• Priorität, Assoziativität, Parameteranzahl: nicht veränderbar
• <b>unäre Postfix-Operator</b> : zusätzlich (Dummy-)Parameter vom Typ int	• <b>unäre Postfix-Operator</b> : zusätzlich (Dummy-)Parameter vom Typ int
Beispiel 1: Überladen als Nichtelementfunktion	Beispiel 1: Überladen als Nichtelementfunktion
meineKlasse operator++(meineKlasse obj&, int) {	meineKlasse operator++(meineKlasse obj&, int) {
meineKlasse alt = obj;	meineKlasse alt = obj;
"++obj"; // „Inkrementieren“ von obj	"++obj"; // „Inkrementieren“ von obj
return alt;	return alt;
Beispiel 2: Überladen als Methode	Beispiel 2: Überladen als Methode
meineKlasse meineKlasse::operator++(int) {	meineKlasse meineKlasse::operator++(int) {
meinTyp alt = *this;	meinTyp alt = *this;
"++(*this)"; // „Inkrementieren“ des Objekts	"++(*this)"; // „Inkrementieren“ des Objekts
return alt;	return alt;
→ ... als Nichtelementfunktion (globale Funktion)	→ ... als Nichtelementfunktion (globale Funktion)
• Alle Operanden werden als Argumente übergeben.	• Alle Operanden werden als Argumente übergeben.
• Mindestens ein Argument muß ein class-Objekt sein.	• Mindestens ein Argument muß ein class-Objekt sein.
<b>retTyp operator&lt;math&gt;(&amp; DeklarListe)</b>	<b>retTyp operator&lt;math&gt;(&amp; DeklarListe)</b>
{ /* Funktionscode */ }	{ /* Funktionscode */ }
• überladene EA-Operatoren	• überladene EA-Operatoren
std::ostream& operator<math><< (\text{std::ostream& os, const \& x}) {	std::ostream& operator<math><< (\text{std::ostream& os, const \& x}) {
// spezifische Anweisungen,	// spezifische Anweisungen,
// z.B. os<<x. getWert1() << ' ' <<x. getWert2();	// z.B. os<<x. getWert1() << ' ' <<x. getWert2();
return os;	return os;
std::istream& operator>> (std::istream& is, T& x) {	std::istream& operator>> (std::istream& is, T& x) {
// spezifische Anweisungen,	// spezifische Anweisungen,
// z.B. int temp; is>>temp; x.setWert (temp);	// z.B. int temp; is>>temp; x.setWert (temp);
return is;	return is;
→ ... als Elementfunktion (Methode)	→ ... als Elementfunktion (Methode)
• Methode darf nicht statisch sein.	• Methode darf nicht statisch sein.
• Das aufrufende Objekt ist automatisch immer der erste Operand	• Das aufrufende Objekt ist automatisch immer der erste Operand
• Argument: evtl. notwendiger zweiter Operand	• Argument: evtl. notwendiger zweiter Operand
• Methode wird vereilt (Ausnahme: Zuweisungsoperator).	• Methode wird vereilt (Ausnahme: Zuweisungsoperator).
<b>retTyp CName::operator&lt;math&gt;(&amp; typ name)</b>	<b>retTyp CName::operator&lt;math&gt;(&amp; typ name)</b>
{ /* Funktionscode */ }	{ /* Funktionscode */ }
• Überladener Zuweisungsoperator:	• Überladener Zuweisungsoperator:
o sollte Wert der Zuweisung zurückgeben (→ Mehrfachzuweisung)	o sollte Wert der Zuweisung zurückgeben (→ Mehrfachzuweisung)
o in polymorphen Klassen oft überschreibbare Methode <b>clone</b>	o in polymorphen Klassen oft überschreibbare Methode <b>clone</b>
• Überladener Zeiger-Selektionsoperator >:	• Überladener Zeiger-Selektionsoperator >:
o muß entweder einen Zeiger auf ein Objekt zurückgeben oder ein Objekt, für das der ->-Operator ebenfalls überladen ist	o muß entweder einen Zeiger auf ein Objekt zurückgeben oder ein Objekt, für das der ->-Operator ebenfalls überladen ist

**Datentyp void**

<b>void</b>	kein Wert ( $\rightarrow$ keine Variablen) ( $\wedge$ kein Typ')
<b>void*</b>	„typenloser“ (d.h. generischer) Zeiger;
<code>void *zeiger=beliebiger zeiger</code>	gestattet, Umkehrung nicht

**Integrale Zahlentypen und Zahlenkonstanten**

<b>Boolesche Typen</b>	
<b>bool</b>	<b>Typ</b> Speicherplatz $\geq 1$ Byte
<b>Boolesche Werte/Litralle</b>	<code>true</code> „Wahr“ ( <code>#0</code> ) <code>false</code> „falsch“ ( <code>=0</code> )
<b>Ganzzahlige Standardtypen mit/ohne Vorzeichen</b>	grundätzlich: <code>sizeof(short)&lt;sizeof(int)&lt;sizeof(long)&lt;sizeof(long long)</code> Eckige Klammern markieren hier optionale Angaben ohne Wirkung.
<b>Typ</b>	<b>Speicherplatz</b> 1 Byte (immer)

**Grenzwerte ganzzahiger Standardtypen <climits>**

<b>CHAR_BIT</b>	8 Anzahl der Bits im kleinsten Nicht-Bitfeld-Typ
<b>MB_LEN_MAX</b>	1 maximale Anzahl von Bytes in Multibyte-Zeichen
• Konkrete Zahlenwerte: Mindestforderungen des C-Standards	
<b>CHAR_MIN</b>	(-127 od. 0) $\leq$ <b>C</b> $\leq$ <b>CHAR_MAX</b> (+127 od. 255)
<b>SCHAR_MIN</b>	(-127) $\leq$ <b>sC</b> $\leq$ <b>SCHAR_MAX</b> (+127)
0	<b>uC</b> $\leq$ <b>UCHAR_MAX</b> (255)
<b>SHRT_MIN</b>	(-32767) $\leq$ <b>sS</b> $\leq$ <b>SHRT_MAX</b> (+32767)
<b>INT_MIN</b>	(-2147483647) $\leq$ <b>iS</b> $\leq$ <b>INT_MAX</b> (2147483647)
<b>LONG_MIN</b>	(-9223372038475807) $\leq$ <b>lUL</b> $\leq$ <b>LONG_MAX</b> (299967295)
0	<b>uUL</b> $\leq$ <b>ULLONG_MAX</b> (-9223372038475807)

**Grenzwerte ganzzahiger Standardtypen <cstdint>**

<b>int_fastN_t</b>	<b>uint_fastN_t</b>	<b>int_leastN_t</b>	<b>uint_leastN_t</b>	schnellster Typ mit <b>WB</b> $\geq N$
<b>intmax_t</b>	<b>uintmax_t</b>			größer ganzzahliger Typ
<b>intptr_t</b>	<b>uintptr_t</b>			optionaler Typ für Wert eines Zeigers
• Konkrete Werte: <code>genau bzw. mindestforderungen des C-Standards</code>				
<b>INTN_MIN</b>	<b>g_(-2^N-1)</b>	<b>INTN_t</b>	<b>INTN_MAX</b>	größte Geltungszahl
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MAX LDBL_MAX
<b>INT_FASTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INT_FASTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
0		<b>uintN_t</b>	<b>UINTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 10 EXP
<b>INTN_MIN</b>	<b>(m_(-2^N-1))</b>	<b>intN_t</b>	<b>INTN_MAX</b>	DBL_MIN 10 EXP LDBL_MIN 1

## Variablen: Deklaration/Initialisierung

- Position im Block beliebig
- Die mehrfache identische Deklaration einer Variablen ist illegal.
- D. im *Init*-Teil der *for*-Schleife mögl., gültig bis Schleifende
- bei Speicherklasse **static**: implizite Initialisierung mit 0-Bytes

Klassische Syntax (mit Konstruktorsyntax mischbar):

```
[Speicherklasse] [Typ-Qualifizierer] typ
name1 [= Ausdruck1] [, name2 [= Ausdruck2] [, ...]];
[Speicherklasse] [Typ-Qualifizierer] typ
name1 [(Konstruktortypargumente1)] [, name2 [(Konstruktortypargumente2)] , ...];
```

**Variablen: Bitfeld** (Komponente von class, struct oder union)

- keine Anwendung des Adreßoperators möglich
- Es dürfen keine Vektoren von Bitfeldern gebildet werden.

```
Bitfeld vom Typ typ mit b Bits,
typ: bool, signed od. unsigned int
```

<b>Nullzeiger</b>	<b>std::nullptr</b>	<b>t nullptr</b>
Dereferenzierung (Zugriff auf Objektwert)	<b>*zname</b>	<b>&amp;zobjname</b>
Adresse des Objekts <b>objname</b>		

Arithmetik **zname+=** bedeutet **zname += sizeof(typ)**

### Referenz

- alternativer Name (Alias) für ein Objekt
- konstanter Zeiger, wird bei Verwendung automatisch dereferenziert  
= L-Value → auf rechter u. linker Seite d. Zuweisung „**=**“ verwendbar
- Die Bindung einer Referenz an eine Variable ist nicht veränderbar.

Deklaration **typ& Referenzname = Variablename;**

## Abgeleitete Datentypen

### Feld (C-Array)

- dim, dimX, zeilen, spalten:** =Anzahl, ganzzahlig;
- Konstante (**const**) oder symbolische Konstante (Makro)
- Bei vorhandener Initialisierung: nichtinitialisierte Feldelemente=0.
- wenn **typ=**=Klassentyp: Aufruf des Default-Konstruktors

```
[Speicherklasse] [Typ-Qualifizierer] typ
name1 [= Ausdruck1] [, name2 [= Ausdruck2] [, ...]];
```

```
[Speicherklasse] [Typ-Qualifizierer] typ
name1 [(Konstruktortypargumente1)] [, name2 [(Konstruktortypargumente2)] , ...];
```

```
Offset (in Byte) einer Komponente (kein Bitfeld) vom Beginn einer Struktur
```

```
size_t offset(sName, komponente)
```

```
<estddef>
```

```
sName;
```

```
Beispiel: struct s1; struct s2;
```

```
(Zeiger erforderlich!)
```

```
s1* s2* name2; ... }
```

```
struct s2 { ...; s1* name1; ... }
```

```
Offset (in Byte) einer Komponente (kein Bitfeld) vom Beginn einer Struktur
```

```
size_t offset(sName, komponente)
```

```
<estddef>
```

```
sName;
```

```
Beispiel: struct s1; struct s2;
```

```
(Zeiger erforderlich!)
```

```
s1* s2* name2; ... }
```

```
struct s2 { ...; s1* name1; ... }
```

## Typ-Informationen, -Definition, -Umwandlung

### • Klasse type info (→ <type info>) Überladen: == und !=

- Name des Typs      const char\* type info::name()
- Typ-Informationen zur Laufzeit:  
const type info& typeid (ausdruck oder typ)

### Typ-Definition

```
Typ auf Basis eines Datentyps definieren typedef dtyp tname
Beispiel: typedef unsigned char uchar;
```

### Konvertierung zwischen elementaren Datentypen

1-dim. <b>[SK] [TQ] typ fname [dim];</b>	<b>[SK] [TQ] typ fname [dim]= [Wert0, Wert1...];</b>
2-dim. <b>[SK] [TQ] typ fname [zeilen][spalten];</b>	<b>[SK] [TQ] typ fname [zeilen][spalten] = [Wertliste Zeile0, Wertliste Zeile1...];</b>
N-dim. <b>[SK] [TQ] typ fname [dim1] [dim2] ... [dimN];</b>	<b>[SK] [TQ] char name[] = "text"</b>

Feld von Zeichen (Ende: '\0') **[TQ] char name[] = "text"**

char x[3] = "abc"; ist ungültig; korrekt ('\'0' !) ist char x[4] = "abc"

### Aufzählung

- automatische Typisierung → **ename** ist somit „echter“ Datentyp.
- keine Angabe **Wert0** → **Wert0=k**. A. **WertN → WertN-WertN-1**
- Integer-kompatibel: enum Var = integer\_ Wert verboten
- etyp**: unterlegter (integraler) Typ der Enumeratoren
- gleichzeitige Variablen Dekl. (**eVar1 usw.**) möglichst vermeiden

**eKey [ename] [: etyp] { Enumerator0 (= Wert0), ... }**

**eKey [evar1, ...];**

### Benutzerdefinierte Konvertierungen

Für benutzerdefinierte Typen können explizite Konvertierungen durch Definition eigener Konvertierungsoperatoren angegeben werden.

class X { private: int z;

public: X(int y) { z=y; }

operator int() { return z; } ...};

### Konvertierung mit Konstruktoren

- Ein Konstruktor (nicht explicit) mit einem einzelnen Parameter kann vom Compiler für implizite Konvertierungen zwischen dem Typ des Parameters und dem Klassentyp verwendet werden.

class X { private: int z;

public: X(int y) { z=y; } ...};

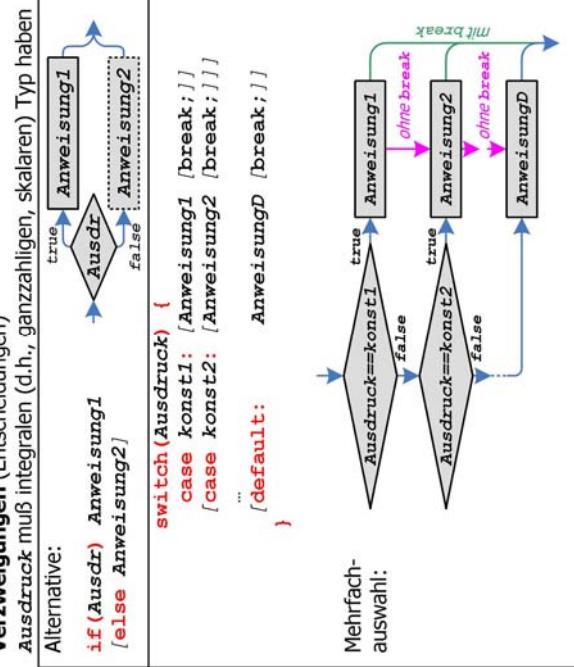
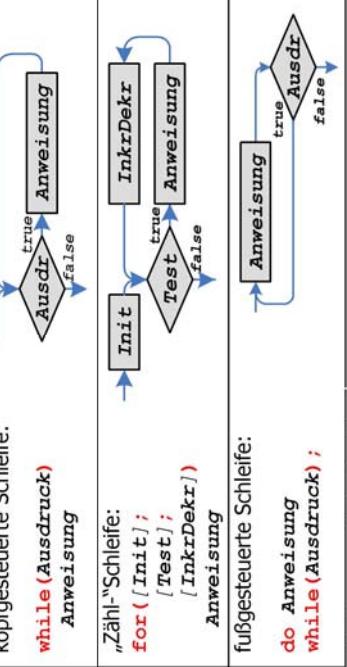
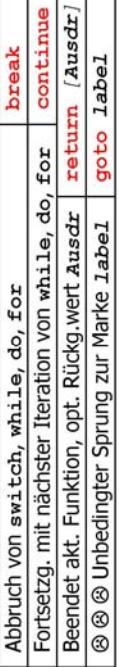
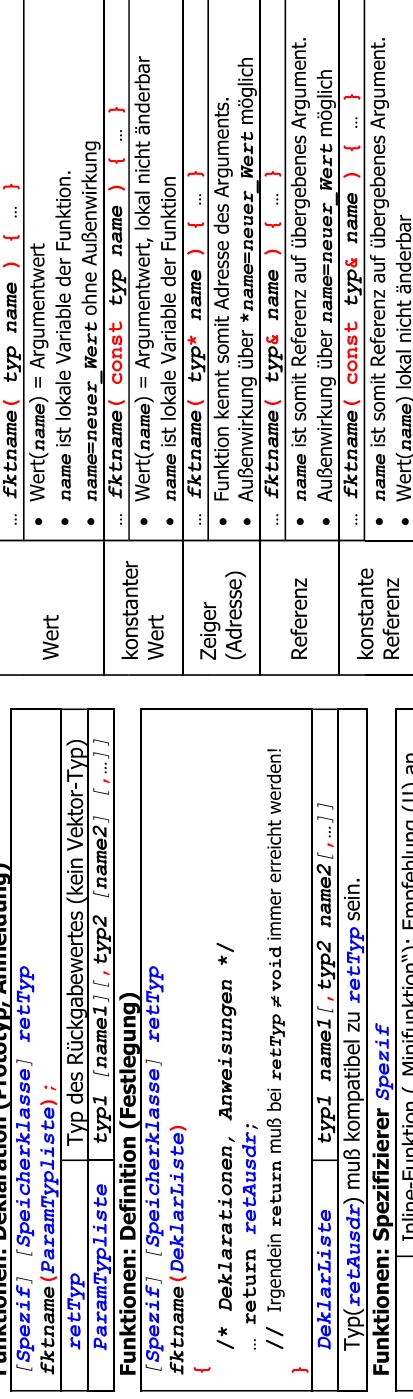
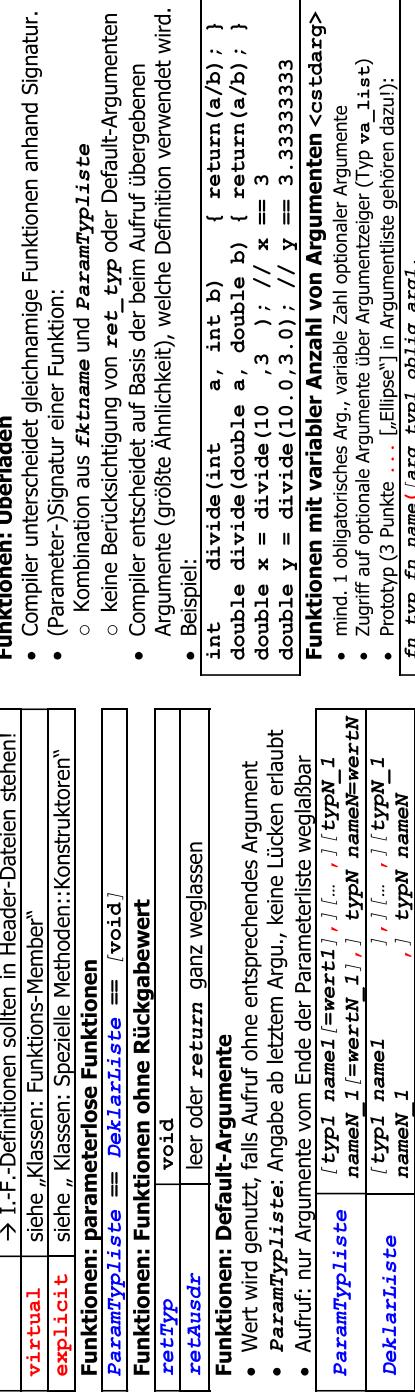
Beispiel: **X x=20; // implizite Konvertierung**

**➔ Vereinigung** (Union, Alternative, Variante, Überlagerung): **C-Sicht**

- Für alle Komponenten wird derselbe Speicherplatz (dieselbe Adresse) in der Größe der größten Komponente reserviert.
- Initialisierung ist auf die erste Variante (Komponente) beschränkt.
- automatische Typisierung → **uName** ist somit „echter“ Datentyp.
- gleichzeitige Variablen Dekl. (**uVar1 usw.**) möglichst vermeiden

**union [uName] { Deklaration\_der\_Komponenten } [uVar1, ...];**

Ver-Variable **uName varname [= {Init\_Komp1}; ... ];**

**Flußkontrolle****Verzweigungen** (Entscheidungen)**Schleifen****Kopfgesteuerte Schleife:****Bedingungsfreie Sprünge****Funktionen****Funktionen: Deklaration (Prototyp, Anmeldung)****Funktionen: Parameterlose Funktionen**

<b>Funktionen: Parameterübergabe</b>	
<b>... mittels</b>	<b>Beschreibung</b> (für Einzelargumente)
Wert	<ul style="list-style-type: none"> <li><code>fktname ( typ name ) { ... }</code></li> <li>• Wert(<code>name</code>) = Argumentwert</li> <li>• <code>name</code> ist lokale Variable der Funktion.</li> </ul>
konstanter Wert	<ul style="list-style-type: none"> <li><code>fktname ( const typ name ) { ... }</code></li> <li>• Wert(<code>name</code>) = Argumentwert, lokal nicht änderbar</li> <li>• <code>name</code> ist lokale Variable der Funktion</li> </ul>
Zeiger (Adresse)	<ul style="list-style-type: none"> <li><code>fktname ( typ* name ) { ... }</code></li> <li>• Funktion kennt somit Adresse des Arguments.</li> <li>• Außenwirkung über <code>*name=newer_Wert</code> möglich</li> </ul>
Referenz	<ul style="list-style-type: none"> <li><code>fktname ( const typ&amp; name ) { ... }</code></li> <li>• <code>name</code> ist somit Referenz auf übergebenes Argument.</li> <li>• Außenwirkung über <code>name=newer_Wert</code> möglich</li> <li>• <code>name</code> ist somit Referenz auf übergebenes Argument.</li> <li>• Wert(<code>name</code>) lokal nicht änderbar</li> </ul>

**Funktionen: Überladen**

Compiler unterscheidet gleichnamige Funktionen anhand Signatur.

(Parameter-)Signatur einer Funktion:

o Kombination aus `fktname` und `ParamTypListe`

o keine Berücksichtigung von `ret_typ` oder Default-Argumenten

• Compiler entscheidet auf Basis der beim Aufruf übergebenen Argumente (gröÙte Ähnlichkeit), welche Definition verwendet wird.

Beispiel:

```
int divide int a, int b) { return(a/b); }
```

```
double divide(double a, double b) { return (a/b); }
```

```
double x = divide(10, 3); // x == 3
```

```
double y = divide(10.0, 3.0); // y == 3.33333333
```

**Funktionen mit variabler Anzahl von Argumenten <cstdarg>**

mind. 1 obligatorisches Arg., variable Zahl optionaler Argumente

• Zugriff auf optionalen Argumenten über Argumentzeiger (`Typ va_list`)

• Prototyp (3 Punkte ... [Ellipse] in Argumentliste gehörten dazu!):

```
fn_name (argTyp1 obligArg1, argTyp2 obligArg2, [usw.])
```

Zeiliger auf Argumente

Initialisierung des übergebenen Arg-zeigers auf 1. optionales Arg.

Zugriff auf aktuelles Argument, Zeiger auf nächstes Arg. setzen

Kopieren eines Objekts vom Typ

va\_list mit aktuellen Status

Beenden Arg-zeiger-Verwendung

`va_end(va_list z_arg)`

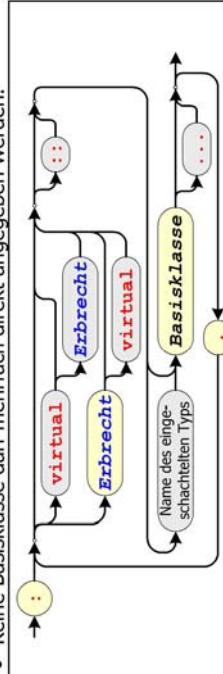
**Klassentypen: Klassen****Klassen: Ankündigung/Vorwärtsdeklaration (für Klasse cname)**`class cname;`**Klassen: Definition (einer Klasse cname)**`class cname { ... };`**Klassen: Methoden (Attribut, Instanzvariable)**`cname::attribute; cname::instanzvariable;`**Klassen: Zugriff (Attribut, Instanzvariable)**`classname::attribute; classname::instanzvariable;`**Klassen: Vererbung**`class derived : public base { ... };`**Klassen: Zugriffsrecht :**`private, protected, public;`**Klassen: Attribut (Deklaration)**`retTyp cname : nameM(Deklarliste);`**Klassen: Methode (Deklaration oder inline-Definition)**`retTyp cname : nameM(wie bei Funktionen);`**Klassen: Friend**`friend class friendname;`**Klassen: Funktionsprototyp**`retTyp cname : nameM(ParamTypListe), const;`**Klassen: Klasse:**`class classname;`**Klassen: eingeschachtelte Typen**`Definition einer Klasse;``Definition einer Aufzählung;``Typedefinition mittels typeid;`**Zugriffsrecht Zugriffsrecht**`Gilt bis zur nächsten Angabe eines Zugriffsrechts.``Voreinstellung: private`**Zugriff erlaubt durch**`Recht Klasse, friend abgeleitete Klasse sonstiges``private ja nein nein ja``protected ja ja ja ja``public ja ja ja ja`**Klassen: Objektdeklaration, Methodenaufru**`Objekt anlegen cname objekt (Starrwerte);``Methodenruf objekt::methode (Parameter)`**Klassen: this-Zeiger**`für jede Klasse automatisch (nur in ihrem Bereich) definierter Zeiger,``der auf die eigene Klasse bzw. auf die aktuelle Instanz verweist.``this Zeiger auf aktuelles Objekt *this das eigene Objekt`**Klassen: Daten-Member (Attribut, Instanzvariable)**`Deklaration [mutable] typ nameAttr;``Zugriff von außen objekt::nameAttr;``objekt::adresse->nameAttr;`**Konstante Daten-Member (konstante Teillobjekte):**`nach Initialisierung nicht mehr veränderbar``Bei der Definition des Konstruktors einer Klasse muß für jedes``konstante Teilobjekt ein Elementinitialisierer angegeben werden.``Zugriff: nur durch nur-lesende Methoden``Deklaration const typ nameConstAttr;`**Elementobjekte**`= Objekt einer anderen Klasse als Attribut``Für jedes E.o. muß ein gleichnamiger Elementobjekt-Konstruktor in``der Initialisierer-Liste angegeben werden.`**statische Daten-Member (Klassenvariablen)**`• gehören zur Klasse an sich, existieren einmal u. objektunabhängig``Deklaration static [const] typKV cname;``Definition [const] typKV cname : nameKV;``Zugriff über Objekt der Klasse cname : nameKV``Klasse (Vorzugsvariante) cname : nameKV``Klassenmethode z.B. get_nameKV(), set_nameKV()`**Klassen: Funktions-Member (Methoden, Elementfunktionen)**`• ähneln stark den → „Funktionen“`**„normale“ Methoden:**`Deklaration retTyp nameM(ParamTypListe);``Definition retTyp cname : nameM(Deklarliste);``{ Anweisungen (wie bei Funktionen) }`**nur-lesende Methoden:**`Deklaration retTyp nameM(ParamTypListe), const;``Definition retTyp cname : nameM(Deklarliste) const;``{ Anweisungen (wie bei Funktionen) }`**virtuelle Methoden (→ Vererbung)**`Deklaration virtual retTyp nameVM(... USW.);``Definition retTyp cname : nameVM(... USW.);`**abstrakte (rein virtuelle) Methoden (→ Vererbung)**`Deklaration virtual retTyp nameAM() =0;``Definition keine Implementierung in der Klasse cname;`**statische Funktions-Member (Klassenmethoden):**`• an kein Objekt der Klasse gebunden; this nicht verfügbar``• nur Zugriff möglich auf Klassenvariablen und -methoden``Deklaration static retTyp nameKM(ParamTypListe);``Definition retTyp cname : nameKM(Deklarliste);``{ Anweisungen (wie bei Funktionen) }``Aufruf über Klasse (bevorzugt zugriffbares Objekt objekt::nameM(ArgListe))`**Klassen: Spezielle Methoden::Konstruktoren**`• Methoden zur Initialisierung einer Instanz einer Klasse``• Konstruktoren können wie Funktionen überladen werden.``nicht erlaubt: virtual, static, const, volatile`**Default-Konstruktor:**`• parameterlos; wird automatisch vom Compiler erzeugt, wenn überhaupt kein einziger Konstruktor in der Klasse angegeben wurde`**Standard-Konstruktor (oft auch „Default-Konstruktor“):**`• parameterlos (oder alle Argumente mit Default-Werten)`**Default-Konstruktor:**`• Voraussetzung für Deklaration eines Felds von Elementen der Klasse``Deklaration /explicit/ cname();``Definition cname :: cname () { Anweisungen }`**parametrisierbarer Konstruktor:**`Deklaration cname (ParamTypListe);``Definition cname :: cname (Deklarliste) { Anweisungen }`**Initialisierer Liste**`Definition cname :: cname (InitialisiererListe) { Anweisungen }`**Spezielle Methoden::Konvertierungsfunctionen**`• zur Konvertierung eines cname-Objekts in Typ zieltyp``Deklaration /explicit/ operator zieltyp() const;``Definition cname : operator zieltyp() const``{ return zieltyp (Umwandlung (*this)); }`**Elementobjekte**`= Objekt einer anderen Klasse als Attribut``Für jedes E.o. muß ein gleichnamiger Elementobjekt-Konstruktor in``der Initialisierer-Liste angegeben werden.`

**Klassen: Freund-Konzept**

- Freunde einer Klasse haben Zugriff auf alle Member der Klasse.
- **Zugriffsrecht** ohne Bedeutung;
- Freundschaft wird nicht ererbt.
- friend-Funktionen-/Methoden: this von CName nicht verfügbar
- friend-Klassen: Alle Methoden des „Freundes“ sind friend-Funktionen der Klasse, die die friend-Deklaration enthält.
- Freunde eingeschachtelter Klassen haben keine besonderen Zugriffsrechte auf Member der umgebenden Klasse.

**→ Zugriffsrechte**

- ohne explicit: virtuelle Basisklasse
- ohne virtual: nicht-virtuelle Basisklasse
- Keine Basisklasse darf mehrfach direkt angegeben werden.

**→ Zugriffsrechte**  
ohne explizites angegebene Ableitungsart **Erbrecht**:

- Kindklasse ist struct: Annahme von public für Erbrecht
- Kindklasse ist class: Annahme von private für Erbrecht

Attribut in der Basisklasse		Erbrecht (Art der Ableitung)	
		public	protected
public	protected	protected	private
protected	protected	protected	private
private		kein Zugriff	

**Klassen: Modulare Programmierung (Vorschlag!)**

```
➔ Klassen: Struktur der Header-Datei k.lassemname.h
#ifndef KLASSENNAME_H_ // Wächter gegen mehrfache Deklaration
#define KLASSENNAME_H_ // mehrfache Deklaration
class KlasseName
{
    // Freunde der Klasse:
    /* Angabe der Freunde der Klasse */
    // Attribute:
    private:
        /* Deklaration der Attribute */
        // Konstruktoren & Co.:
    public:
        KlasseName() ; // Default-Konstruktor
        KlasseName(const KlasseName& c) ; // Copy-Konstr.
        KlasseName& operator=(const KlasseName& c);
        virtual ~KlasseName() ; // Destruktor
    // Operationen:
    public:
        /* Deklaration/Inline-Implementierung von Methoden, die ein
         * Anwender an Objekten dieser Klasse aufrufen kann. U.a.
         * get_...() - und set_...() -Methoden für jedes Attribut */
        // überladbare Methoden (overridables):
    public:
        /* Deklaration/Inline-Implementierung von Methoden, die in einer
         * Unterklasse dieser Klasse überladen werden können oder sollen.
         * Diese müssen virtual sein. */
    private:
        /* Deklaration/Inline-Implementierung von Methoden, die lediglich
         * intern zur Implementierung bestimmter Funktionen dienen. */
};

#endif // Ende des Wächters KLASSENNAME.H_
```

**➔ Klassen: Struktur der Quellcode-Datei klasseName.cpp**

```
#include "klassenname.h"
/* Klassenvariablen: Definition+Initialisierung */
/* Implementierung aller deklarierten Methoden */
```

**Klassentypen: Strukturen**

- funktional identisch zu Klassen, einzige Unterschiede:
  - **struct** anstelle von **class**
  - Default-Zugriffsrecht: public
- Unterschiede zu Klassen:
  - können immer nur einen Wert für einen Datenmember auf einmal aufnehmen
  - **union** anstelle von **class**
  - Default-Zugriffsrecht: public
  - keine virtual-Methoden
  - keine Vererbung (weder von noch an etwas)
  - Member: Keine Objekte, die Konstruktoren oder Destruktoren definieren oder den Zuweisungsoperator überladen.
  - Unions können anonym (unbenannt) sein.

**Klassentypen: Vereinigungen**

- Unterschiede zu Klassen:
  - können immer nur einen Wert für einen Datenmember auf einmal aufnehmen
  - **union** anstelle von **class**
  - Default-Zugriffsrecht: public
  - keine virtual-Methoden
  - keine Vererbung (weder von noch an etwas)
  - Member: Keine Objekte, die Konstruktoren oder Destruktoren definieren oder den Zuweisungsoperator überladen.
  - Unions können anonym (unbenannt) sein.

**Dynamische Speicherverwaltung** (Kurzfassung)

- Freigabe nicht vergessen!!!
- `xalloc()`  $\Leftrightarrow$  `delete` und `new`  $\Leftrightarrow$  `free()` nicht kombinieren!!!
- `newArg`: Argumente bei überladenen `new`-Operator

Anforderung (Allocierung):

```
typ* [::] new [newArg] typ [Konstruktorkonstante] ]
ditto für Feld (Für typ muß der Standard-Konstruktor existieren):
typ* new [newArg] typ [anzahl]
Freigabe (Typ von zeig: typ*): void [::] delete [::] zeig
ditto für Feld:
void [::] delete [] zeig
```

**Ausnahmebehandlung** (Exception Handling) (Kurzfassung)

```
try {
    ... // zu überwachende kritische Anweisungen
    [throw Ausnahme;] // Auslösen der Ausnahme
    ...
} catch(Typ_der_Ausnahme [&] Ausnahme) {
    ... // Anweisungen für Fehlerbehandlung
    [throw;] // Weiterreichen der Ausnahme
    ...
}
// beliebig viele weitere catch-Blocke möglich
catch (...) { // Auffangen aller Ausnahmen
    ... // Anweisungen für Fehlerbehandlung
}
```

**Schablonen (Templates)** (Kurzfassung)

- = Vorlagen zur parametrisierten Erzeugung neuer Versionen einer Funktion oder Klasse ( $\rightarrow$  Spezialisierung)

**Template-Klassen**

Deklaration:

```
template <Deklarliste>
class CName { wie normale Klassendefinition };
Instantiiierung CName<Argumentliste> ObjektName;
```

Beispiel

```
template <class T, int a>
class Paar {
private: T x, y; T feld[a];
public: Paar(T a, T b):x(a),y(b) {}
    T getMax();
    ... // irgendwas sinnvolles mit dem feld
};

template <class T, int a>
T Paar<T, a>::getMax() { return x>y?x:y; }

int main() {
    Paar<int, 5> meinPaar(80, 45);
    cout << meinPaar.getMax();
}
```

**Template-Funktionen**

Deklaration:

```
template <Deklarliste>
retTyp fname (ParamTypliste);
```

Beispiel

```
template <class T>
T groesser(T a, T b) {
    return (a>b?a:b);
}

int main() {
    int a=9, b=2, c;
    float x=5.3, y=3.2, z;
    c=groesser(a,b);
    z=groesser(x,y);
}
```

**Kleine Sammlung nützlicher Dinge****Bitbeeinflussung**

n-tes Bit in x :	Setzen x  = (1<<n)	Invertieren x ^= (1<<n)
	Löschen x &= ~(1<<n)	Prüfen (gesetzt?) x & (1<<n)

**Felder**

Anzahl der Feldelemente (geht nur im Geltungsbereich /ok/!!):	anzahl = sizeof(feldname)/sizeof(feldname[0])	
Arbeit mit der Standardeingabe		
→ Leren des Tastaturpuffers	#define TP_LEREEN { cin.clear(); \ cin.ignore(INT_MAX, '\n'); }	
→ Warten auf Eingabe von Enter	#define WARTEN_AUF_ENTER { TP_LEREEN; cin.get(); }	

**Notizen**