

Sprache C++

Erläuterungen

<code>xyz</code>	terminales „xyz“ (unverändert zeichenweise zu übernehmen)
<code>abc</code>	metasprachl. Symbol: für „abc“ Einsetzen der Definition von <code>abc</code>
\doteq	Definitionszeichen. (ggf. auch innerh. einer Alternative) – dagegen terminales Gleichheitsz.: =
<code> </code>	metasprachl. Alternativ-Zeichen – dagegen terminaler senkrechter Strich:
<code>{ }</code>	metasprachl. Klammer – dagegen terminale eckige Klammern: []
<code>abc_{0..n}</code>	null oder mehrere Male „abc“, Folge beliebig vieler „abc“
<code>abc_{1..n}</code>	ein oder mehrere Male „abc“, Folge mit mindestens einem „abc“
<code>abc_{opt}</code>	optionales „abc“ (gleiche Bedeutung wie: <code>abc_{0..1}</code>)
$\left[\begin{array}{l} abc \\ def \end{array} \right]$	gleichwertig zu: $\left[abc \mid def \right]$ [NV] Definition nicht vollständig
<code>Op2</code> ^{Op2}	Bezug auf Operator(-Hierarchiestufe) [NE+1] nicht erlaubt (in dieser Vorlesung)
¹⁰	Bezug auf Syntaxelement <Text> Fachausdruck in englisch
	[...] Erläuterung (nicht zur Syntax gehör.)

Sprachhinweise:

(*nichts*) in C und C++

`C++` nur in C++

`C` nur in C

`C/C++` in C; auch in C++ möglich und üblich (meist als Gegensatz zu `C++` gebraucht)

`C (C++)` in C; in C++ zwar auch möglich, aber nicht empfohlen: dort bessere Konstrukte vorhanden

`C++(neu)` C++, neu, ggf. noch nicht in gängigen Compilern implementiert

`C++(alt)` C++, alt, ggf. gerade noch in gängigen Compilern implementiert

1 Syntax (Auswahl)

1.1 Gesamte Auswahl

$xxx^{-1}LISTE \doteq xxx \left[, xxx \right]_{0..n}$

Übersetzungseinheit (Quelldatei) \doteq ¹⁰*Deklaration*_{0..n}

¹⁰*Deklaration* _[NV] \doteq

¹¹*EinfacheDeklaration* | ³³*FunktionsDefinition* | `C++(neu)` ⁴⁰*NamensbereichDefinition*

¹¹*EinfacheDeklaration* \doteq

¹²*DeklSpezifizierer*_{0..n} $\left[\begin{array}{l} \sup{30} \text{ Deklarator} \\ \text{[Dekl.-Punkt]} \\ \text{Initialisierer}_{opt} \end{array} \right]^{-1}LISTE_{opt} ;$

¹²*DeklSpezifizierer* \doteq

¹³*SpeicherklassenSpezifizierer* | ¹⁴*TypSpezifizierer* | `typedef` |

`C++` ¹⁵*FunktionsSpezifizierer* | `C++` *friend*

¹³*SpeicherklassenSpezifizierer* \doteq `auto` | `register` | `static` | `extern` | `C++(neu)` *mutable*

¹⁴*TypSpezifizierer* _[NV] \doteq

`char` | `short` | `int` | `long` | `signed` | `unsigned` | `float` | `double` | `void` |

`C++(neu)` `bool` | ²¹*KlassenSpezifizierer* | ²³*EnumSpezifizierer* | ^{32a}_{opt} ²⁰*TypName* |

²⁵*CvQualifizierer*

¹⁵*FunktionsSpezifizierer* `C++` \doteq `inline` | `virtual` | `C++(neu)` *explicit*

²⁰*TypName* \doteq *KlassenName* | *EnumName* | *TypedefName*

²¹*KlassenSpezifizierer* _[NV] \doteq

`C/C++` $\left[\begin{array}{l} \text{struct} \\ \text{union} \end{array} \right] \text{Bezeichner}_{opt} \left\{ \text{ElementDeklaration}_{0..n} \right\} |$

`C++` $\left[\begin{array}{l} \text{class} \\ \text{struct} \\ \text{union} \end{array} \right] \text{KlassenName}_{opt} \left[: \text{BasisSpezifizierer}^{-1}LISTE \right]_{opt} \\ \left\{ \left[\begin{array}{l} \sup{22} \text{ ZugriffsSpezifizierer} : \\ \text{ElementDeklaration} \end{array} \right]_{0..n} \right\}$

²²*ZugriffsSpezifizierer* `C++` \doteq `private` | `protected` | `public`

²³*EnumSpezifizierer* \doteq `enum` *EnumName*_{opt} $\left\{ \begin{array}{l} \sup{24} \text{ EnumeratorDefinitions} \\ \sup{24} \text{ EnumeratorDefinitions}^{-1}LISTE_{opt} \end{array} \right\}$

²⁴*EnumeratorDefinition* \doteq *Enumerator* $\left[= \text{KonstanterAusdruckIntegralenTyps} \right]_{opt}$

²⁵*CvQualifizierer* \doteq `const` | `volatile`

³⁰*Deklarator* \doteq ³²*DirekterDeklarator* | ³¹*Zeigeroperator* ³⁰*Deklarator*

³¹*Zeigeroperator* _[NV] \doteq `*` ²⁵*CvQualifizierer*_{0..n} | `C++` `&`

³²*DirekterDeklarator* _[NV] \doteq

^{32a}_{opt} *Bezeichner* | $\left(\begin{array}{l} \sup{30} \text{ Deklarator} \end{array} \right) | \begin{array}{l} \sup{32} \text{ DirekterDeklarator} \\ \left[\text{KonstanterAusdruck}_{opt} \right] \end{array} |$

$\boxed{\text{c/c++}}$ ³² *DirekterDeklarator* (*ParameterDeclarations*⁻¹ *LISTE*_{opt}) |
 $\boxed{\text{C++}}$ ³² *DirekterDeklarator* (*ParameterDeclarations*⁻¹ *LISTE*_{opt}) ²⁵ *CvQualifizierer*_{0..n} |
 $\boxed{\text{C++}}$ ^{32a} \otimes_{opt} *OperatorFunktionsName* \doteq **operator** *Operator* |
 $\boxed{\text{C++}}$ ^{32a} \otimes_{opt} *KonversionsFunktionsName* \doteq **operator** ¹⁴ *TypSpezifizierer*_{1..n} ³¹ *Zeigeroperator*_{0..n} |
 $\boxed{\text{C++}}$ ^{32a} \otimes_{opt} \sim *KlassenName*
^{32a} \otimes (*EingebetteterNamensSpezifizierer*) _[NV] \doteq [*KlassenName* ::]_{1..n}
³³ *FunktionsDefinition* \doteq
 $\boxed{\text{c/c++}}$ ¹² *DeklSpezifizierer*_{0..n} ³⁰ *Deklarator* ⁵⁴ *ZusammengesetzteAnweisung* |
 $\boxed{\text{C++}}$ ¹² *DeklSpezifizierer*_{0..n} ³⁰ *Deklarator* [: *ElementInitialisierer*⁻¹ *LISTE*]_{opt} ⁵⁴ *ZusammengesetzteAnweisung*
⁴⁰ *NamensbereichDefinition* $\boxed{\text{C++(neu)}}$ \doteq **namespace** *NamensbereichName*_{opt} { ¹⁰ *Deklaration*_{0..n} }
⁵⁰ *Anweisung* \doteq
⁵¹ *AusdrucksAnweisung* | ⁵² *MarkierteAnweisung* | ⁵³ *SprungAnweisung* |
⁵⁴ *ZusammengesetzteAnweisung* | ⁵⁵ *AuswahlAnweisung* | ⁵⁶ *WiederholungsAnweisung* |
 $\boxed{\text{C++}}$ ⁵⁷ *DeklarationsAnweisung*
⁵¹ *AusdrucksAnweisung* \doteq *Ausdruck*_{opt} ;
⁵² *MarkierteAnweisung* \doteq
case *KonstanterAusdruck* : ⁵⁰ *Anweisung* | **default** : ⁵⁰ *Anweisung* |
_[NEP1] *Bezeichner* : ⁵⁰ *Anweisung*
⁵³ *SprungAnweisung* \doteq
break ; | **continue** ; | **return** *Ausdruck*_{opt} ; | _[NEP1] **goto** *Bezeichner* ;
⁵⁴ *ZusammengesetzteAnweisung* (*VerbundAnweisung*, *Block*) \doteq
 $\boxed{\text{C++}}$ { ⁵⁰ *Anweisung*_{0..n} } | $\boxed{\text{c/c++}}$ { ¹¹ *EinfacheDeklaration*_{0..n} ⁵⁰ *Anweisung*_{0..n} }
⁵⁵ *AuswahlAnweisung* \doteq ⁶⁰ *if-Anweisung* | ⁶¹ *switch-Anweisung*
⁵⁶ *WiederholungsAnweisung* \doteq ⁷⁰ *while-Anweisung* | ⁷¹ *do-while-Anweisung* | ⁷² *for-Anweisung*
⁵⁷ *DeklarationsAnweisung* $\boxed{\text{C++}}$ _[NV] \doteq ¹¹ *EinfacheDeklaration*
⁶⁰ *if-Anweisung* \doteq **if** (⁹⁹ *Bedingung*) ⁵⁰ *Anweisung* [**else** ⁵⁰ *Anweisung*]_{opt}
⁶¹ *switch-Anweisung* \doteq **switch** (⁹⁹ *Bedingung*) *Anweisung*
Anm.: Anweisung i. a. als ⁵⁴ *ZusammengesetzteAnweisung*, *darin mehrmals* ⁵² *MarkierteAnweisung*
⁷⁰ *while-Anweisung* \doteq **while** (⁹⁹ *Bedingung*) ⁵⁰ *Anweisung*
⁷¹ *do-while-Anweisung* \doteq **do** ⁵⁰ *Anweisung* **while** (*Ausdruck*) ;
⁷² *for-Anweisung* \doteq
 $\boxed{\text{c/c++}}$ **for** (*Ausdruck1*_{opt} ; *Ausdruck2*_{opt} ; *Ausdruck3*_{opt}) ⁵⁰ *Anweisung* |
 $\boxed{\text{C++}}$ **for** (¹¹ *EinfacheDeklaration1* ⁹⁹ *Bedingung2*_{opt} ; *Ausdruck3*_{opt}) ⁵⁰ *Anweisung*
Zusammengefaßt $\boxed{\text{C++}}$ (*Anm.: auch für EinfacheDeklaration gilt bei* $\boxed{\text{C++(neu)}}$ *Anm. zu Nummer* ⁹⁹):
for ([⁵¹ *Ausdrucksanw1*] ¹¹ *EinfacheDeklaration1*) ⁹⁹ *Beding2*_{opt} ; *Ausdr3*_{opt}) ⁵⁰ *Anweisung*
Anm.: 1 Initialisierung, 2 Schleifeneintrittsbedingung (fehlend: true), 3 Reinitialisierung
⁹⁹ *Bedingung* \doteq *Ausdruck* | $\boxed{\text{C++(neu)}}$ ¹⁴ *TypSpezifizierer*_{1..n} ³⁰ *Deklarator* = *Ausdruck*
Anm.: 2. Alternative $\boxed{\text{C++(neu)}}$: *Variablen-Gültigkeitsber. nur innerhalb der Kontrollstruktur (Anweisung* ^{60,61,70,72}).

1.2 Auswahl für 1. Semester

xxx⁻¹ *LISTE* \doteq *xxx* [, *xxx*]_{0..n}

Übersetzungseinheit (*Quelldatei*) \doteq ¹⁰ *Deklaration*_{0..n}

¹⁰ *Deklaration* \doteq
¹¹ *EinfacheDeklaration* | ³³ *FunktionsDefinition* | $\boxed{\text{C++(neu)}}$ ⁴⁰ *NamensbereichDefinition*

¹¹ *EinfacheDeklaration* \doteq
¹² *DeklSpezifizierer*_{0..n} [³² *DirekterDeklarator* _[Dekl.-Punkt] *Initialisierer*_{opt}]⁻¹ *LISTE*_{opt} ;

¹² *DeklSpezifizierer* \doteq
char | **short** | **int** | **long** | **signed** | **unsigned** | **float** | **double** | **void** |
bool | *TypedefName* | **typedef**

- ³²*DirekterDeklarator* \doteq
 $Bezeichner \mid (\text{ } ^{32}DirekterDeklarator \text{ }) \mid \text{ } ^{32}DirekterDeklarator [KonstanterAusdruck_{opt}] \mid$
 $\text{ } ^{32}DirekterDeklarator (ParameterDeklarations\text{-}^1LISTE_{opt})$
- ³³*FunktionsDefinition* \doteq
 $\text{ } ^{12}DeklSpezifizierer_{0..n} \text{ } ^{32}DirekterDeklarator \text{ } ^{54}ZusammengesetzteAnweisung$
- ⁴⁰*NamensbereichDefinition* $\boxed{C++(neu)}$ \doteq namespace NamensbereichName_{opt} { ¹⁰*Deklaration*_{0..n} }
- ⁵⁰*Anweisung* \doteq
 $\text{ } ^{51}AusdrucksAnweisung \mid \text{ } ^{52}MarkierteAnweisung \mid \text{ } ^{53}SprungAnweisung \mid$
 $\text{ } ^{54}ZusammengesetzteAnweisung \mid \text{ } ^{55}AuswahlAnweisung \mid \text{ } ^{56}WiederholungsAnweisung \mid$
 $\text{ } ^{57}DeklarationsAnweisung$
- ⁵¹*AusdrucksAnweisung* \doteq Ausdruck_{opt} ;
- ⁵²*MarkierteAnweisung* \doteq case KonstanterAusdruck : ⁵⁰*Anweisung* | default : ⁵⁰*Anweisung*
- ⁵³*SprungAnweisung* \doteq break ; | continue ; | return Ausdruck_{opt} ;
- ⁵⁴*ZusammengesetzteAnweisung (VerbundAnweisung, Block)* \doteq { ⁵⁰*Anweisung*_{0..n} }
- ⁵⁵*AuswahlAnweisung* \doteq ⁶⁰*if-Anweisung* | ⁶¹*switch-Anweisung*
- ⁵⁶*WiederholungsAnweisung* \doteq ⁷⁰*while-Anweisung* | ⁷¹*do-while-Anweisung* | ⁷²*for-Anweisung*
- ⁵⁷*DeklarationsAnweisung* \doteq ¹¹*EinfacheDeklaration*
- ⁶⁰*if-Anweisung* \doteq if (Ausdruck) ⁵⁰*Anweisung* [else ⁵⁰*Anweisung*]_{opt}
- ⁶¹*switch-Anweisung* \doteq switch (Ausdruck) ⁵⁴*ZusammengesetzteAnweisung*
- ⁷⁰*while-Anweisung* \doteq while (Ausdruck) ⁵⁰*Anweisung*
- ⁷¹*do-while-Anweisung* \doteq do ⁵⁰*Anweisung* while (Ausdruck) ;
- ⁷²*for-Anweisung* \doteq for (Ausdruck1_{opt} ; Ausdruck2_{opt} ; Ausdruck3_{opt}) ⁵⁰*Anweisung*
 Anm.: 1 Initialisierung, 2 Schleifeneintrittsbedingung (fehlend: true), 3 Reinitialisierung

2 Operatoren Stufen 1 bis 17: fallende Hierarchie Assoziativität: \rightarrow ; Ausnahmen („@“): \leftarrow

1	a	::	$\boxed{C++}$	global	6	ab	+ -	Addit., Subtrakt.
	b	::	$\boxed{C++}$	Bereichsauflösung	7	ab	<< >>	Bitverschiebung
2	a	[]		Index	8	abcd	< <= > >=	Vergleich
	b	()		Funktionsaufruf	9	ab	== !=	(Un-)Gleichheit
	c	()	$\boxed{C++}$	Umw. EinfTypName	10		&	bitw. Und
	de	. ->		Elementzugriff	11		^	bitw. Exkl.-Oder
	fg	++ --		Postfix-Inkrement/Dekrem.	12			bitw. Inkl.-Oder
	h-k	art_cast<>()	$\boxed{C++(neu)}$	Typumw., s. u.	13		&&	log. Und
	l	typeid	$\boxed{C++(neu)}$	Typidentifizierung	14			log. (Inkl.-)Oder
3@	ab	++ --		Präfix-Inkrement/Dekrem.	15@		? :	Bedingungsoperator
unär	c	~		Bitinversion	ternär			
präfix	d	!		log. Negation	16@	a	=	(einf.) Zuweisung
	ef	+ -		Vorzeichen		bcde	*= /= %= +=	zusges. Zuweisung
	g	&		Adresse		fgh	-- >>= <<=	
	h	*		Dereferenzierung		ijk	&= ^= =	
	i	()		Typumwandlung	17		,	Kommaoperator
	j	sizeof		Speichergröße				
	kl	new delete	$\boxed{C++}$	Freispeicher				
4	ab	->* .*	$\boxed{C++}$	Elementzugriff				
5	a	*		Multiplikation				
	b	/		Division				
	c	%		Teilerrest				

Nicht überladbar: :: . sizeof .* ?:
 Zu Op2h-k $\boxed{C++(neu)}$: art_cast<>() Typumw.:
 art \doteq static | dynamic | const | reinterpret

3 Standard-Streams $\boxed{C++}$

Anm. für $\boxed{C (C++)}$: Bibliotheksfunktions-Familien aus <stdio>

Ausgabe: printf/puts/putchar, Eingabe: scanf/gets/getchar

3.1 Spezielle Ausdrucksanweisungen (<iostream>)

– Wert der Gesamtausdrücke (Haupteffekt): Streamobjekt-Referenz (s. Anm.) –

Schreiben in Standardausgabeeinheit \doteq `cout` [<< *Ausdruck*]_{0..n} ;

Schreiben in Fehlerausgabeeinheit \doteq [`cerr` | `clog`] [<< *Ausdruck*]_{0..n} ;

(in UNIX umleitbar, in DOS nicht) Anm.: `C++` `cerr`, `C++(neu)` `clog`

Einlesen aus Standardeingabeeinheit \doteq `cin` [>> *Variable*]_{0..n} ;

Einlesen mit Operator >>: Überlesen führende ZwrZeichen (Zwischenraumzeichen <white-spaces>): Leerz, Tab, CR, LF, NeueSeite, vertTab), Lesen bis ausschließlich nächstes unpassendes Zeichen (bei String: ZwrZeichen)

– dagegen Lesen aller Zeichen s. (3.3)

Anzahl der derzeit im Puffer von `cin` vorhandenen Zeichen: `cin.rdbuf()->in_avail()`

Anmerk.: Streamobjekte (z. B. `cin`, `cout`) oder deren Referenzen im Falle Boolescher Wertung:

`C++(neu)` `false` bei Fehler/Dateiende, sonst `true` (bzw. `C++(alt)` Zeiger 0, sonst Zeiger!=0);
im Falle `false` (0): vor Weiterbearbeitung unbedingt Elementfunktion `clear()` aufrufen!

3.2 Formatierungen (i. w. für die Ausgabe)

Alle aufgeführten Setzungen bleiben erhalten bis zur nächsten Änderung (Ausnahme: `setw/width`)

<u>Manipulator</u>	<u>Elementfunktion</u>	
<code>endl</code>	—	NeueZeile+PufferLeer
<code>ends</code>	—	Nullbyte+PufferLeer
<code>flush</code>	<code>flush</code> [⊗]	PufferLeer
<code>dec</code> , <code>oct</code> , <code>hex</code>	—	Setzen Basis für Ganzzahlen (Std.: <code>dec</code>)
<code>setfill(int)</code> [⊙]	<code>fill(char)</code> ^{△▽}	Füllzeichen (Standard: '␣')
<code>setw(int)</code> [⊙]	<code>width(int)</code> ^{△▽}	Mindestausgabebreite; automat. Rücksetzen auf Standard 0
<code>setprecision(int)</code> [⊙]	<code>precision(int)</code> ^{△▽}	Standard: 6
bei <code>ios::scientific/fixed</code> gesetzt:		Anzahl Nachkommastellen
beide nicht gesetzt		Anzahl führende Stellen insgesamt

– Weitere Manipulatoren s. (3.4): gleichlautend mit Flags, zusätzlich teilweise auch Verneinung –

Manipulation der Flags, siehe (3.4)

<code>setiosflags(long int)</code> [⊙]	<code>setf(long int)</code> [▽]	Setzen nur der 1-Bit-Stellen
—	<code>setf(wert,maske)</code> [▽]	Ändern der 1-Bit-Stellen aus <code>maske</code> in Werte aus <code>wert</code> (beide <code>long int</code> ; s. a. (3.4))
<code>resetiosflags(long int)</code> [⊙]	<code>unsetf(long int)</code> [▽]	Löschen nur der 1-Bit-Stellen
—	<code>flags(long int)</code> ^{△▽}	vollständiges Übernehmen der Bits

Anmerk.: [⊙] <iomanip>, alle anderen: <iostream>; [△] Aufruf ohne Parameter: Rückgabe aktueller Wert;
[▽] Aufruf mit Parameter: Setzen neuen Wert, Rückgabe alter Wert; [⊗] Rückgabe Streamobjekt-Ref.

3.3 Spezielle Funktionen (als Elementfunktionen) – für Lesen aller Zeichen, auch ZwrZeichen – Rückgabewert der folgenden drei Funktionen: Streamobjekt-Referenz, s. (3.1 Anmerk.)

<code>get(char &zeich)</code>	Einlesen Zeichen auf <code>zeich</code> – auch Umlaute/„ß“
<code>getline(char *ziel,int anz,char ende='\n')</code>	
<code>get(char *ziel,int anz,char ende='\n')</code>	[⚠] keine aufeinanderfolgende Aufrufe dieses <code>get</code> ! Einlesen Zeichenkette auf <code>ziel</code> , garantierter Nullbyteabschluss; Lesen maximal <code>anz-1</code> Zeichen, weniger, wenn vorher <code>ende</code> oder EOF; <code>ende</code> -Zeichen wird gelesen (bei <code>getline</code>) bzw. nicht gelesen (bei <code>get</code>); keinesfalls wird <code>ende</code> -Zeichen nach <code>ziel</code> kopiert.
<code>get()</code>	[⚠] Rückgabe Zeichen Typ <code>int</code> , EOF bei Fehler/Dateiende; Vorsicht bei Umlauten/„ß“!!

3.4 Wichtige Flags (Aufzählungskonstanten der Klasse `ios`)

Alle Setzungen bleiben wirksam bis nächste Änderung; Bit-Inklusiv-Oder-Verknüpfung^{Op12} der Flags möglich
`right*`, `left*`, `internal*` Positionierung rechtsbündig (Std.), linksbündig,

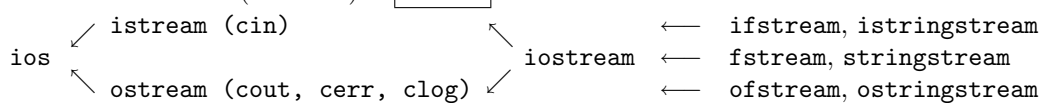
	Setzen der Füllzeichen zw. Vorzeichen u. Zahl; Maske <code>adjustfield</code> [▽]
<code>dec*</code> , <code>oct*</code> , <code>hex*</code>	Basis Ganzzahlen, Std.: <code>dec</code> ; Maske <code>basefield</code> [▽]
<code>scientific*</code> , <code>fixed*</code>	Gleitkomma-, Festkommadarstellung; Std.: keines; Maske <code>floatfield</code> [▽]
<code>showbase</code> [⊗]	Basis bei Ganzzahlen ausgeben (Präfix in C/C++-Schreibw.)
<code>showpoint</code> [⊗]	schließende Nullen (sowieso bei <code>scientific/fixed</code>)
<code>showpos</code> [⊗]	auch Pluszeichen ausgeben bei positivem Wert
<code>uppercase</code> [⊗]	Großbuchst. bei Hex.-/Exp.-Darstellung (sonst klein)
<code>boolalpha</code> [⊗]	<code>C++(neu)</code> Boolesche Werte symbolisch (Std.: 0, 1) – auch für Eingabe
<code>skipws</code> [⊗]	ZwrZeichen <white spaces> überlesen (Std.)

Anmerk.: * max. jeweils 1 Bit gesetzt; auch als Manipulator (3.2), dann implizites Löschen konkurrierender

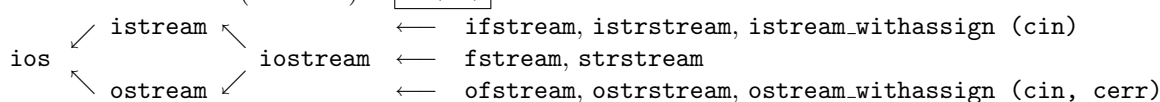
Flags; \otimes auch als Manipulator (3.2), zusätzl. Manipulatorform `no...` (Löschen), beides ggf. `C++(neu)`; ∇ Mas-
kenname (Klasse `ios`) für `setf`-Überladung mit Par. `maske` (3.2)

4 Allgemeines über Streams `C++`

Klassenhierarchie (Auswahl)



Klassenhierarchie (Auswahl)



Auswahl *Elementfunktionen* und *Aufzählungskonstanten* (alle Konstant. aus `ios`, d. h. `ios::Konst.`)
Deklarationen in `<iostream>`, Dateihandhabung in `<fstream>`, Param. nicht immer angegeben

Klasse `ios`:

Streamzustand: `goodbit`, `eofbit`¹⁾, `failbit`²⁾, `badbit`³⁾
log.: `good`, `eof`, `fail` (`failbit` \vee `badbit`), `bad`; Rücksetz.: `clear`⁴⁾, Bits: `rdstate`
log. Abfrage (`StreamObj`) bzw. (`!StreamObj`) wirkt wie `!fail` bzw. `fail`

Formatierung und weitere Flags s. (3.2), (3.4)

Klasse `istream`: \bullet `getline`⁵⁾, \bullet `putback(char)`⁶⁾, \bullet `ignore(anz=1, ende=EOF)`⁷⁾, `peek`⁸⁾
 \bullet `operator>>`⁹⁾, \bullet `get`⁵⁾, \bullet `read(char*, size_t)`¹⁰⁾, `tellg`¹¹⁾, \bullet `seekg`¹¹⁾

Klasse `ostream`: \bullet `operator<<`⁹⁾, \bullet `put`, \bullet `write(char*, size_t)`¹⁰⁾, `tellp`¹¹⁾, \bullet `seekp`¹¹⁾,
 \bullet `flush`

Klassen `ifstream`, `ofstream`, `fstream`:

Dateimodus(`ios`): `in` (Std. bei `istream`), `out` (Std. bei `ostream`),
`ate`¹²⁾, `app`¹³⁾, `trunc`¹⁴⁾, `binary` (Binärmodus; Std.: Textmodus)
`open(Dateiname)`¹⁵⁾, `open(Dateiname, Modus)`¹⁵⁾, `close`¹⁶⁾

Anmerkungen: \bullet Rückgabe Streamobjekt-Referenz (außer bei `get`, wenn ohne Par.), s. (3.3); ¹⁾ gleich-
zeitig wird `failbit` gesetzt; ²⁾ letzte Op. nicht erfolgreich, Forts. nach `clear` wohl mögl.; ³⁾ Stream defekt,
Forts. kaum sinnvoll; ⁴⁾ ohne Par. Löschen aller Fehlerbits (NÖTIG nach Fehler), sonst mit Par. (`neuBit-`
`Status`); ⁵⁾ s. (3.3); ⁶⁾ dazu auch `C++(neu)` \bullet `unget` oh. Par.: letztes geles. Zeichen; ⁷⁾ auch `ende`-Zeichen
(Par.-Typ `int`) wird (ggf.) verworfen; ⁸⁾ Typ `int` (wie `get` oh. Par., s. (3.3)), Zeichen bleibt im Stream; ⁹⁾ mit
vielen Überladungen; ¹⁰⁾ auch gut geeignet für binäre Dateien (Flag `binary` setzen); ¹¹⁾ `...p` (`put`), `...g` (`get`),
`streampos tell...` Rückgabe akt. Ort, `seek...` (`tell...`-Wert) absolut positionieren, `seek...` (`Offset, Bezug`) rela-
tiv positionieren mit `Bezug beg, cur, end` (`beg`: wie abs. Positionieren), Positionszeiger `...g` und `...p` sind iden-
tisch bei gleichem Puffer; ¹²⁾ nach Öffnen an Dateiende („at end“) setzen; ¹³⁾ Schreiben nur am Dateiende;
¹⁴⁾ löscht Datei bei `out` \wedge \neg (`ate` \vee `app`); ¹⁵⁾ Parameter beider `open`-Überladungen auch schon bei Konstruktor
möglich, ggf. jedoch nicht bei `fstream`; ¹⁶⁾ auch implizit durch Destruktor

5 Gültigkeitsbereich, Speicherklasse, Bindung

5.1 Gültigkeitsbereich (scope) für Namen:

- lokal (Name heißt lokal/intern): von Dekl.-Punkt (innerh. Block) bis zugehör. Blockende,
⁵⁵ *Auswahl*-, ⁵⁶ *Wiederholungsanw.* mit ⁹⁹ *Bedingung*: `C++(neu)` gesamte Anweisung implizit geblockt
- global (Name heißt global/extern): von Dekl.-Punkt (außerh. Block) bis Programmdateiende,
`C++` Zugriff trotz Verdeckung möglich mit unärem Präfixoperator `::Op1`
- Funktion: nur bei Marken⁵² `[NEF]`
- Klasse: Klassenelemente

Wichtig: Ein Name wird durch Neudeklaration desselben Namens in untergeordn. Blöcken verdeckt.
Anm.: Name Aufzählungs-/Klassentyp und Name Fkt./Var. in selber Gültigkeitsbereichsebene möglich; Aufz.-
/Kl.-Typname ist bei Überlappung verdeckt, jedoch mit `enum/class/struct/union Typname` erreichbar (`C`):
immer nur so, da „Etikett“ (tag) mit eigenem Namensraum).

5.2 Speicherklasse (storage class): statisch, automatisch; dazu auch dyn. kontrolliert (`new`, `delete`)

– NICHT zu verwechseln mit ¹³ *Speicherklassenspezifizierer* –

	globaler Name	lokaler Name - mit <code>static</code> ¹⁾ -	lokaler Name - ohne <code>static</code> ²⁾ -
Speicherklasse Lebensdauer impliz. Initialis. expliz. Initialis. - wie oft	statisch Programm ja (0 bzw. ³⁾) mit konst. Ausdr. ⁴⁾ einmal		automatisch Block nein bzw. ³⁾ mit belieb. Ausdr. jedesmal
Gültigkeitsber.	global	lokal	lokal

Anmerkungen:
 1) oder auch **extern**, dann aber i. a. nur Deklaration
 2) fehlend oder **auto** oder **register**
 3) bei Objekten: Standardkonstr.
 4) `C++(neu)` auch nichtkonstant

5.3 Bindung <linkage>: extern, intern, keine

Dekl.-Spezifiz.	globaler Name	lokaler Name
keiner	externe B. ¹⁾	keine B.
extern <code>const_{opt}</code>	externe B. ²⁾	externe B. ³⁾
static	interne B.	keine B.
<code>C++</code> inline	interne B.	—
const (oh. extern)	interne B.	keine B.

Anm.: 1) Funktionsdef., -dekl., Variablendef.
 2) Deklaration (bei Var. mit Initial.: Def.)
 3) nur Deklaration;
 falls Name bekannt, Übername Bindung
 Hinw.: • **typedef**-Name immer ohne Bindung
 • `C++` **ClassName** immer immer ext. Bindung

6 Überladung von Operatoren `C++`

Operatorart	Ausdruck	Globale Funktion äquivalenter Ausdruck Funktionsdeklaration	Elementfunktion äquivalenter Ausdruck Funktionsdeklaration
binär	$a \otimes b$	$\text{operator} \otimes (a, b)$ $Te \text{ operator} \otimes (Ta \text{ para}, Tb \text{ parb});$	$a.\text{operator} \otimes (b)$ $Te \text{ operator} \otimes (Tb \text{ parb});$
unär präfix (Op3)	$\otimes a$	$\text{operator} \otimes (a)$ $Te \text{ operator} \otimes (Ta \text{ para});$	$a.\text{operator} \otimes ()$ $Te \text{ operator} \otimes ();$
Inkr./Dekr. postfix	$a \otimes$	$\text{operator} \otimes (a, 1)$ $Te \text{ operator} \otimes (Ta \text{ para}, \text{int});$	$a.\text{operator} \otimes (1)$ $Te \text{ operator} \otimes (\text{int});$

a, b Ausdrücke, Ta, Tb zugehörige Typen, $para, parb$ Parameter, Te Ergebnistyp, \otimes Operatorzeichen
 Anm.: Überladung unärer Postfixoperatoren (Op2) siehe C++-Bücher

7 Typumwandlungen

7.1 Arithmetische Umwandlungen <arithmetic conversions>

Umwandlung bei vielen binären Operatoren, wenn Operanden verschiedenen Typs. Abkürzung (hier):
 $Umw(T) \doteq$ WENN ein Operand vom Typ T , DANN Umwandl. and. Operand \rightarrow Typ T , Ergebnis Typ T , ENDE
 SONST weiter

Umwandlungsregeln:

- $Umw(\text{long double})$ (7.4)
- $Umw(\text{double})$ (7.4)
- $Umw(\text{float})$ (7.4)
- Ganzzahl-Erweiterung (7.2) auf beide Operanden
- $Umw(\text{unsigned long int})$ (7.3)
- WENN `long int` u. `unsigned int` vorhand.:
 WENN $W_{\text{unsigned int}} \subseteq W_{\text{long int}}$ DANN
`unsigned int` \rightarrow `long int` (7.3)
 SONST beide \rightarrow `unsigned long` (7.3)
- $Umw(\text{long int})$ (7.3)
- $Umw(\text{unsigned int})$ (7.3)
- - jetzt nur noch `int` vorhanden -

7.2 Ganzzahlige Typangleichung, Ganzzahl-Erweiterung <integral promotion>

Umwandlung des Quelltyps (`char/short/Ganzzahl-Bit-Feld`, bei `C` `C++(alt)` Aufzählungstyp):

WENN $W_{\text{Quelltyp}} \subseteq W_{\text{int}}$ DANN \rightarrow `int` SONST \rightarrow `unsigned int` (7.3)

`C++(neu)` bei `bool`: `false` \rightarrow 0, `true` \rightarrow 1

`C++(neu)` bei Aufz.-Typ: \rightarrow Typ mit pass. Werteber.: `int, unsigned int, long int, unsigned long int`

7.3 Ganzzahlige Typumwandlung <integral conversion>

Ganzzahl-Wert \rightarrow `unsigned`-Ganzzahl-Wert: Umwandlung in Wert mod 2^n (n Anz. Bits des neuen Typs)
 Ganzzahl-Wert \rightarrow `signed`-Ganzzahl-Wert: Beibehaltung Wert, wenn möglich, sonst impl.-def. Verhalten

7.4 Gleitkomma-Umwandlungen

Gleitkomma-Typ → Ganzzahl-Typ: Abschneiden gebroch. Anteil; wenn Wert nicht darstellbar, Verh. undef.

Ganzzahl-Typ → Gleitkomma-Typ: Wenn Wert darstellbar, Auf-/Abrundung*), sonst undefiniert

Gleitkomma-Typ → genauerer Gleitkomma-Typ: Wert bleibt erhalten

Gleitkomma-Typ → ungenauerer Gleitkomma-Typ: Wenn Wert darstellbar, Auf-/Abrund.*), sonst undef.

*) nur wenn nötig; hierbei jedoch keine allg. Festlegung, ob Auf- oder Abrundung

7.5 Typ von Ganzzahl-Konstanten

Konst. hat jeweils den ersten Typ, in dem Wert darstellbar ist:

- Konstante ohne Suffix: `int`, (nur oktal/hex. :) `unsigned int`, `long int`, `unsigned long int`
- Konstante mit Suffix `U` oder `u`: `unsigned int`, `unsigned long int`
- Konstante mit Suffix `L` oder `l`: `long int`, `unsigned long int`