

# Kurzreferenz C++

## Programmstruktur

Kommentare bis Zeilenende	<i>/* Kommentar */</i> <i>// Kommentar</i>
Deklarationen	<i>typen, konstanten, funktionen</i>
Hauptprogramm oder	<code>int main(int argc, char* argv[])</code> <code>int main()</code> <code>{anweisungen}</code>

## Funktionen

anmelden (Kopf)	<code>Typ f(parameter);</code> <code>auto f(parameter) -&gt;Typ;</code>
optimierbar	<code>inline</code>
festlegen (Rumpf)	<code>Typ f(parameter){anweisungen}</code>
Parameterliste	<code>Typ name, ...</code>
Vorgabewert	<code>Typ name=wert</code>
Funktion aus	<code>auto f=</code>
Lambda-Ausdruck	<code>[einschlussliste](parameter)</code> <code>-&gt;Typ<sub>opt</sub> {anweisungen};</code>
aufrufen	<code>f(argumente);</code> <code>auto ergebnis=f(argumente);</code>

## Programmaufteilung

Headerdatei *.h	<code>#ifndef headername</code>
mit Wächter	<code>#define headername</code>
gegen doppelte	<code>deklarationen</code>
Deklaration	<code>#endif</code>
Implementierung *.cpp / *.cc	<code>#include "headername"</code> <i>weitere includes</i>
Programmteil	<i>Funktionen</i>
Namensraum	<code>namespace bereich {...}</code>
Alias	<code>namespace name = bereich;</code>
importieren	<code>using namespace bereich;</code>

## Präprozessor

Einbinden	
Bibliothek	<code>#include &lt;datei&gt;</code>
eigene Datei	<code>#include "datei"</code>
Makro	<code>#define name text</code>
-Funktion	<code>#define name(var) text</code>
Beispiel	<code>#define abs(x) \</code>
mit Folgezeile	<code>(- (x) &lt; (x)) ? (x) : - (x)</code>
löschen	<code>#undef name</code>
Zeichenkette	<code>#x</code>
Verschmelzen	<code>a##b</code>
bedingte	<code>#if bedingung</code>
Übersetzung	<code>#elif</code>
(optional)	<code>#else</code>
(zwingend)	<code>#endif</code>
<code>#ifdef x</code> für	<code>#if defined(x)</code>
<code>#ifndef x</code>	<code>#if !defined(x)</code>

## Ablaufsteuerung

Anweisung	<i>ausdruck;</i>
Anweisungsblock	<code>{anweisungen}</code>
(void)Funktion verlassen	<code>return;</code>
mit Rückgabewert	<code>return ergebnis;</code>
Sprung	
aus Schleifenblock-Ende	<code>continue;</code>
aus Schleife / switch	<code>break;</code>
innerhalb einer Funktion	<code>goto marke;</code>
Sprungziel	<code>marke:</code>

## Wiederholungen (Schleifen)

über Wertfolge	<code>for(Typ element: liste) anweisung</code>
Zählschleife	<code>for(init; bedingung; schritt)</code> <code>anweisung</code>
kopfgesteuert	<code>while(bedingung) anweisung</code>
fußgesteuert	<code>do anweisung</code> <code>while(bedingung);</code>

## Entscheidungen

einfach	<code>if(init; opt bedingung) anweisung<sub>1</sub></code>
kann entfallen	<code>else<sub>opt</sub> anweisung<sub>2</sub></code>
mehrfach	<code>switch(init; opt ausdruck) {</code>
Durchläufer	<code>case wert<sub>1</sub>: [[fallthrough]]</code>
jeder Fall	<code>case wert<sub>2</sub>: anweisungen</code>
mit Abschluss	<code>break;</code>
sonst-Zweig	<code>default: anweisungen</code> <code>}</code>

## Zusicherungen, Ausnahmebehandlung

Abschalten der	<code>#define NDEBUG</code>
Laufzeittests	<code>#include &lt;cassert&gt;</code>
Zusicherung	<code>assert(test);</code>
bei Übersetzung	<code>static_assert(test, meldung<sub>opt</sub>);</code>
Ausnahme werfen	<code>throw ausdruck;</code>
weiterwerfen	<code>throw; (in catch-Block)</code>
wird erwartet	<code>try {anweisungen}</code>
fangen	<code>catch(Typ&amp; ausnahme)</code>
behandeln	<code>{anweisungen}</code>
alle anderen	<code>catch(...){anweisungen}</code>

## Konstanten (Literale)

Wahrheitswerte	<code>false true</code>
Ganzzahlen	<code>0 1 -1234 1'234 12L 12U</code>
binär / oktal	<code>0b10'1010 0377</code>
hexadezimal	<code>0xFFFF</code>
Gleitkommazahl	<code>1.0 -0.9 3e8 1.6e-19</code>
einfach genau	<code>3.14f</code>
Einzelzeichen	<code>'A' 'z' '0'</code>
Zeile, Tab, ...	<code>'\n' '\t' '\r' '\'' '\\"'</code>
oktal, hex, utf8	<code>'\101' '\xFF' u8'ä'</code>
Zeichenkette utf8	<code>"Hallo" u8"Welt"</code>

## Typen

hergeleitet / ohne	auto void
logisch, Zeichen	bool char wchar_t
ganzzahlig	short int long
Modifizierer	signed unsigned
nichtganzzahlig	float double
Umbenennung	using <i>neuename</i> = <i>Typ</i> ; typedef <i>Typ neuename</i> ;
Aufzählung	enum class <sub>opt</sub> <i>Typ</i> : <i>Basis</i> <sub>opt</sub> { <i>name</i> = <i>wert</i> <sub>opt</sub> ,...};
Überlagerung	union <i>Typ</i> { <i>komponenten</i> };

## Klassen, Strukturen

Ankündigung	struct <i>Typ</i> ; class <i>Typ</i> ;
Definition	class <i>Typ</i> {
Zugriffsrechte	public/private/protected:
Folge beliebig	<i>methoden, attribute</i>
	};
Zugriff erlaubt	friend <i>funktion/klasse</i>
klassenbezogen	static <i>methode/variable</i>
Attribut	<i>Typ name</i> = <i>wert</i> <sub>opt</sub> ;
Methodenkopf	<i>Typ f</i> ( <i>parameter</i> )
Modifizierer	const overload final
polymorph	virtual <i>virtuelleMethode</i> virtual <i>abstrakteMethode</i> =0
Destruktor	virtual <sub>opt</sub> ~ <i>Typ</i> ()=default;
Konstruktor	explicit <sub>opt</sub> <i>Typ</i> ( <i>parameter</i> )
Kopie, verschieben	<i>Typ</i> ( <i>Typ const</i> & <i>x</i> ) <i>Typ</i> ( <i>Typ</i> && <i>x</i> )
Zuweisung	<i>Typ</i> & operator=( <i>Typ</i> & <i>x</i> )
Vererbung	class <i>Abgeleitet</i>
abgeleitet von	: <i>art Basis</i> , ... {
überschreiben	<i>zu ändernde methoden</i>
ergänzen	<i>zusatzkomponenten</i>
	};
Vererbungsart	public/protected/private
bei Rhombus	virtual <i>Basis</i>
Implementierung	<i>typ Typ</i> : : <i>f</i> ( <i>parameter</i> )
Methode	{ <i>anweisungen</i> }
Konstruktor	<i>Typ</i> : : <i>Typ</i> ( <i>parameter</i> )
Initialisiererliste	: <i>Basis</i> { <i>wert</i> }, <i>attribut</i> { <i>wert</i> } { <i>anweisungen</i> }
Destruktor	<i>Typ</i> : : ~ <i>Typ</i> () { <i>anweisungen</i> }
Objektzeiger	this (in <i>Methode</i> )
Objekt anlegen	<i>Typ objekt</i> { <i>wertliste</i> };
auch	<i>Typ objekt</i> ; <i>Typ objekt</i> ( <i>werte</i> );
Methodenaufruf	<i>objekt</i> . <i>f</i> ( <i>argumente</i> )

## Schablonen

Funktion	template<typename <i>T</i> ...> <i>fkt</i>
Struktur, Klasse	template<typename <i>T</i> ...> <i>Typ</i>
spezialisiert	<i>Typ</i> < <i>T</i> ...> <i>fkt</i> < <i>T</i> ...>( ... )

## Variablen

Variable (mit Anfangswert)	<i>Typ name</i> = <i>wert</i> <sub>opt</sub>
Anfangswertliste bei Strukturen	= <sub>opt</sub> { <i>wert</i> , ...}
aus Struktur/Feld	auto[ <i>x,y,z</i> ]=...
Feld (Reihe, array)	<i>Typ name</i> [ <i>n</i> ]
mit <i>n</i> Werten	={ <i>wert</i> <sub>0</sub> , ...}
mehrdimensional	<i>name</i> [ <i>m</i> ][ <i>n</i> ]...
Zeichenkette	char <i>s</i> []="az"
Referenz	<i>Typ</i> & <i>ref</i> = <i>alias</i>
Zeiger	<i>Typ</i> * <i>ptr</i> = <i>adresse</i>
nur lesbar	<i>Typ</i> const ...
in const-Methode änderbar	mutable ...
beim Übersetzen berechenbar	constexpr ...
statisch / lokale Bindung	static ...
flüchtig, nicht optimieren	volatile ...

## Operatoren (nach Rang geordnet)

Namensbereich-Auflösung	<i>bereich</i> : : <i>name</i>
Funktionsaufruf	<i>funktion</i> ()
Feldzugriff	<i>feld</i> [ <i>index</i> ]
Struktur-Komponente	<i>objekt</i> . <i>teil</i>
Zugriff über Zeiger	<i>ptr</i> -> <i>teil</i>
Erhöhen, Absenken um 1 nach /	<i>x</i> ++ <i>x</i> --
vor Auswertung	++ <i>x</i> -- <i>x</i>
Vorzeichen	+ <i>x</i> - <i>x</i>
logisches NICHT	! <i>x</i> not <i>x</i>
bitweises NICHT	~ <i>x</i> compl <i>x</i>
Zeigerinhalt, Adresse	* <i>ptr</i> & <i>objekt</i>
Speicher anfordern (Einzelobjekt)	new <i>Typ</i> { <i>args</i> } <sub>opt</sub>
Feld	new <i>Typ</i> [ <i>n</i> ]
freigeben (Einzelobjekt/Feld)	delete [ <i>opt ptr</i>
Speicherbedarf in Byte	sizeof( <i>name</i> )
Typecast	( <i>Typ</i> ) <i>ausdruck</i>
Komponentenauswahl	<i>objekt</i> . * <i>kptr</i>
über Objektzeiger	<i>ptr</i> ->* <i>kptr</i>
Punktrechnung, Divisionsrest	<i>x</i> * <i>y</i> <i>x</i> / <i>y</i> <i>m</i> % <i>n</i>
Strichrechnung	<i>x</i> + <i>y</i> <i>x</i> - <i>y</i>
Bits um <i>n</i> Stellen schieben	<i>m</i> << <i>n</i> <i>m</i> >> <i>n</i>
kleiner (oder gleich)	<i>x</i> < <i>y</i> <i>x</i> <= <i>y</i>
größer (oder gleich)	<i>x</i> > <i>y</i> <i>x</i> >= <i>y</i>
gleich / ungleich	<i>x</i> == <i>y</i> <i>x</i> != <i>y</i>
bitweises UND	<i>x</i> & <i>y</i> <i>x</i> bitand <i>y</i>
bitweises Exklusiv-ODER	<i>x</i> ^ <i>y</i> <i>x</i> xor <i>y</i>
bitweises ODER	<i>x</i>   <i>y</i> <i>x</i> bitor <i>y</i>
logisches UND	<i>x</i> && <i>y</i> <i>x</i> and <i>y</i>
logisches ODER	<i>x</i>    <i>y</i> <i>x</i> or <i>y</i>
bedingter Ausdruck	<i>bed</i> ? <i>dann</i> : <i>sonst</i>
Ausnahme werfen	throw <i>ausdruck</i>
Zuweisung und Kurzschrift	<i>x</i> = <i>wert</i>
für + - * / % << >> &   ^	<i>x</i> += <i>y</i> für <i>x</i> = <i>x</i> + <i>y</i>
Liste von Ausdrücken	,

Einstellige Operatoren/Zuweisungen binden von rechts.

# Standard-Bibliothek (Auswahl)

in namespace `std`

## Container

### Allgemeine Container-Eigenschaften

Initialisierung	$C\{wertliste\}$
Kopie	$C(C_2)$
aus Bereich	$C(f,l)$
Zuweisung	$C=C_2$
Tausch	$C.swap(C_2)$
Vergleich	$== !=$
lexikographisch	$< <= >= >$
ist leer?	$C.empty()$
Anzahl Werte	$C.size()$
max. Anzahl	$C.max_size()$
Iteratoren	$C.begin() C.end()$
lesend	$C.cbegin() C.cend()$
rückwärts	$C.rbegin() C.rend()$
	$C.crbegin() C.crend()$
Einfügen	$C.insert(pos, x)$
Entfernen	$C.erase(pos)$
	$C.erase(f, l)$
	$C.clear()$

### Assoziative Container

<code>unordered_</code> <code>multi</code>	<code>set&lt;T&gt;</code>
<code>unordered_</code> <code>multi</code>	<code>map&lt;Key, Value&gt;</code>
Vergleich Werte	$C.value\_comp()$
Schlüssel	$C.key\_comp()$
Zählen	$C.count(key)$
Suchen	$C.find(key)$
Anfang	$C.lower\_bound(key)$
Ende	$C.upper\_bound(key)$
Bereich	$C.equal\_range(key)$
Einfügen	$C.insert(x)$
Bereich	$C.insert(f, l)$
Entfernen	$C.erase(key)$

### Mengen <set> <unordered\_set>

geordnet	<code>multi</code> <code>set&lt;T&lt;H&gt;</code>
ungeordnet	<code>unordered_</code> <code>multi</code> <code>set&lt;T&lt;H&gt;</code>
Kriterium <code>&lt;</code> <code>H</code>	<code>less&lt;T&gt;</code> <code>hash&lt;T&gt;</code>

### Assoziative Felder <map> <unordered\_map>

Schlüssel/Wert	<code>multi</code> <code>map&lt;K, V&lt;H&gt;</code>
	<code>unordered_</code> <code>multi</code> <code>map&lt;K, V&lt;H&gt;</code>
Eintrag	<code>pair&lt;const K, V&gt;</code>
Kriterium <code>&lt;</code> <code>H</code>	<code>less&lt;K&gt;</code> <code>hash&lt;K&gt;</code>
Wert-Zugriff	$C.at(k)$ $C[k]$ $C[k]=v$

<code>array&lt;T, 5&gt;</code>	<code>[1 2 3 4 5]</code>
<code>vector&lt;T&gt;</code>	<code>[1 2 3 4 5] &lt;-&gt;</code>
<code>deque&lt;T&gt;</code>	<code>&lt;-&gt; [1 2 3 4 5] &lt;-&gt;</code>
<code>list&lt;T&gt;</code>	<code>[1]-[2]-[3]-[4]-[5]</code>
<code>forward_list&lt;T&gt;</code>	<code>[1]-&gt;[2]-&gt;[3]-&gt;</code>
<code>set&lt;T&gt;</code>	<code>{1 2 3 4 5}</code>
<code>multiset&lt;T&gt;</code>	<code>{1 2 3 3 5}</code>
<code>map&lt;K, V&gt;</code>	Mueller : 3373721 Schulze : 4632536

### Sequentielle Container

<code>vector&lt;T&gt;</code> <code>deque&lt;T&gt;</code> <code>list&lt;T&gt;</code> <code>forward_list&lt;T&gt;</code>	
Konstruktoren	$C(n)$ $C(n, x)$
Zuweisung	
$n$ Std-Werte	$C.assign(n)$
$n$ mal Wert $x$	$C.assign(n, x)$
Bereich	$C.assign(f, l)$
erstes Element	$C.front()$
letztes Element	$C.back()$
Einfügen bei $pos$	$C.insert(pos)$
$n$ mal Wert $x$	$C.insert(pos, n, x)$
Bereich	$C.insert(pos, f, l)$
am Ende	$C.push\_back(x)$
auf $n$ Elemente	$C.resize(n)$
mit $x$ auffüllen	$C.resize(n, x)$
Entferne hinten	$C.pop\_back()$

### Feldcontainer <array> <vector>

Anzahl $N$ fest	<code>array&lt;T, N&gt;</code>
dynamisch	<code>vector&lt;T&gt;</code>
Feldzugriff	$C[index]$ $C.at(index)$

### doppelendige Schlange <deque>

wie <code>vector&lt;T&gt;</code>	<i>zusätzlich</i>
Einfügen vorn	$C.push\_front(x)$
Entfernen vorn	$C.pop\_front()$

### Listen <list> <forward\_list>

Einfügen vorn	$C.push\_front(x)$
Einspleißen	$C.splice(pos, list)$
ab $start$	$C.splice(pos, list, start)$
Bereich $f, l$	$C.splice(pos, list, f, l)$
Einmischen	$C.merge(list<H>)$
Sortieren	$C.sort(<H>)$
Umdrehen	$C.reverse()$
Entfernen vorn	$C.pop\_front()$
	$C.remove(wert)$
Zutreffen $P$	$C.remove\_if(P)$
Dubletten	$C.unique(P_2)$
<code>forward_list&lt;T&gt;</code>	$C.before\_begin()$ $...\_end()$ $C.splice\_after(pos, list...)$

## Algorithmen <algorithm>

### nicht modifizierend

Anwenden $F$	<code>for_each(f, l, F)</code>
Quantoren	<code>all_of(f, l, P)</code> <code>any_of(f, l, P)</code> <code>none_of(f, l, P)</code>
Zählen $wert$ Zutreffen $P$	<code>count(f, l, wert)</code> <code>..._if(f, l, P)</code>
Suche nach $wert$ Zutreffen $P$ Wert $\in [f_2, l_2)$ Schluss $[f_2, l_2)$ Anfang $[f_2, l_2)$ $n$ malig Nachbarn	<code>find(f, l, wert)</code> <code>..._if(f, l, P)</code> <code>..._first_of(f, l, f_2, l_2, P_2)</code> <code>..._end(f, l, f_2, l_2, P_2)</code> <code>search(f, l, f_2, l_2, P_2)</code> <code>..._n(f, l, n, wert, P_2)</code> <code>adjacent_find(f, l, P_2)</code>
Binärsuche $wert$ Grenzen	<code>binary_search(f, l, wert, &lt;)</code> <code>lower_bound(f, l, wert, &lt;)</code> <code>upper_bound(f, l, wert, &lt;)</code>
Bereich	<code>equal_range(f, l, wert, &lt;)</code>
Minimum	<code>min(a, b, &lt;)</code>
Maximum	<code>max(a, b, &lt;)</code>
im Bereich (Position)	<code>min_element(f, l, &lt;)</code> <code>max_element(f, l, &lt;)</code>
Eingrenzen	<code>clamp(x, lo, hi, &lt;)</code>
Vergleich	<code>equal(f, l, f_2, l_2, P_2)</code>
Sortierfolge $[f, l] < [f_2, l_2)$	<code>lexicographical_compare(f, l, f_2, l_2, &lt;)</code>
Unterschied ab	<code>mismatch(f, l, f_2, l_2, P_2)</code>

### modifizierend (wertändernd)

Tauschen	<code>swap(a, b)</code>
Kopieren	<code>copy(f, l, to)</code> <code>..._backward(f, l, to)</code>
Ausfüllen	<code>fill(f, l, wert)</code> <code>..._n(f, n, wert)</code>
mit Funktor	<code>generate(f, l, Gen)</code> <code>..._n(f, n, Gen)</code>
Ersetzen	<code>replace(f, l, alt, neu)</code> <code>..._if(f, l, P, neu)</code> <code>..._copy(f, l, to, alt, neu)</code> <code>..._copy_if(f, l, to, P, neu)</code>
Entfernen	<code>remove(f, l, wert)</code> <code>..._if(f, l, P)</code> <code>..._copy(f, l, to, wert)</code> <code>..._copy_if(f, l, to, P)</code>
ohne Dubletten	<code>unique(f, l, P_2)</code> <code>..._copy(f, l, to, P_2)</code>
Umrechnen	<code>transform(f, l, to, F)</code> <code>transform(f, l, f_2, l_2, to, ⊕)</code>

## mutierend (Reihenfolge ändernd)

Umkehren	<code>reverse(f, l)</code> <code>..._copy(f, l, to)</code>
Teile tauschen	<code>rotate(f, mitte, l)</code> <code>..._copy(f, mitte, l)</code>
Durchmischen $n$ Werte	<code>shuffle(f, l, RandGen)</code> <code>sample(f, l, to, n, RandGen)</code>
Permutieren	<code>next_permutation(f, l, &lt;)</code> <code>prev_permutation(f, l, &lt;)</code>
Sortieren	<code>sort(f, l, &lt;)</code> <code>stable_sort(f, l, &lt;)</code> <code>partial_sort(f, mitte, l, &lt;)</code> <code>..._copy(f, l, f_2, l_2, &lt;)</code>
Teilbereich bis zum $n$ -ten	<code>nth_element(f, nth, l, &lt;)</code>
Zweiteilen bzgl. $P$	<code>partition(f, l, P)</code> <code>stable_partition(f, l, P)</code>
Mischen sortiert	<code>merge(f, l, f_2, l_2, to, &lt;)</code> <code>inplace_...(f, mitte, l, &lt;)</code>
$[f, l] \supseteq [f_2, l_2)?$	<code>includes(f, l, f_2, l_2, &lt;)</code>
$[f, l] \cup [f_2, l_2)$	<code>set_union...</code>
$[f, l] \cap [f_2, l_2)$	<code>set_intersection...</code>
$[f, l] \setminus [f_2, l_2)$	<code>set_difference...</code>
$[f, l] \Delta [f_2, l_2)$	<code>set_symmetric_difference(f, l, f_2, l_2, to, &lt;)</code>

## numerische Algorithmen <numeric>

ggT / kgV	<code>gcd(m, n)</code> <code>lcm(m, n)</code>
Folge	<code>iota(f, l, init)</code>
Summe	<code>accumulate(f, l, init, ⊕)</code> <code>reduce(f, l, init, ⊕)</code> von $F(x_i)$ <code>transform_reduce(f, l, init, ⊕, F)</code>
Skalarprodukt	<code>transform_reduce(f, l, f_2, init, ⊕, ⊙)</code> <code>inner_product(f, l, f_2, init, ⊕, ⊙)</code>
Nachbardifferenz	<code>adjacent_difference(f, l, to, ⊖)</code>
Teilsummen	<code>partial_sum(f, l, to, ⊕)</code> mit/ohne letztem Wert von $F(x_i)$ <code>inclusive_scan(f, l, to, ⊕, init)</code> <code>exclusive_scan(f, l, to, init, ⊕)</code> <code>transform_inclusive_scan(f, l, to, ⊕, fn, init)</code> <code>transform_exclusive_scan(f, l, to, init, ⊕, F)</code>
Bereiche $[first, last)$	$f, l$ $f_2, l_2$
Zielbereichanfang Iterator	$to$
Funktion $f(x)$	$F$
Generator $x=Gen()$	$Gen$ $RandGen$
Prädikat ein-/ <code>bool P(x)</code>	$P$
zweistellig <code>bool P_2(x, y)</code>	$P_2$
Vergleich <code>bool &lt;(x, y)</code>	$x < y$
Binäroperator $x \text{ op } y$	$x \oplus y$ $x \odot y$ $x \ominus y$
optionales Argument, z.B. $<$	$<$

## Zubehör

### Container-Adapter <stack> <queue>

ist leer?	<code>a.empty()</code>
Anzahl Elemente	<code>a.size()</code>
Vergleiche	<code>&lt; == ...</code>

### Stapel `stack<T, Copt>`

Einfügen Wert $x$	<code>st.push(x)</code>
Entfernen (ohne Rückgabe)	<code>st.pop()</code>
oberstes Element	<code>st.top()</code>

### Warteschlange `queue<T, Copt>`

Einfügen Wert $x$	<code>q.push(x)</code>
Entfernen (ohne Rückgabe)	<code>q.pop()</code>
erstes Element	<code>q.front()</code>
letztes Element	<code>q.back()</code>

... zum Vordrängeln `priority_queue<T, Copt, <>>`

Sortierkriterium `<` `less<T>`

Einfügen Wert $x$	<code>pq.push(x)</code>
Entfernen (ohne Rückgabe)	<code>pq.pop()</code>
oberstes Element	<code>pq.top()</code>

mit Funktionen aus `<algorithm>`

Herstellen Heap	<code>make_heap(f, l, &lt;&gt;)</code>
* $(l-1)$ dazu	<code>push_heap(f, l, &lt;&gt;)</code>
* $f$ nach hinten	<code>pop_heap(f, l, &lt;&gt;)</code>
Heap-Sort	<code>sort_heap(f, l, &lt;&gt;)</code>

## Wrapper

Tupel <tuple>	<code>tuple&lt;Typliste&gt;</code> <code>make_tuple(param)</code>
Zugriff	<code>t.get&lt;Typ&gt;()</code> <code>t.get&lt;nr&gt;()</code>
geordnetes Paar	<code>pair&lt;U, V&gt;</code>
<utility>	<code>p.first</code> <code>p.second</code>
Vergleich	<code>== &lt; ...</code>
evtl. vorhanden	<code>optional&lt;T&gt;</code>
<optional>	<code>o.has_value()</code> <code>o.value_or(y)</code>
<variant>	<code>variant&lt;Typliste&gt;</code>
bel. Typ <any>	<code>any</code>
Smarte Zeiger	<code>unique_ptr&lt;T&gt;</code> <code>shared_ptr&lt;T&gt;</code>
<memory>	<code>make_unique&lt;T&gt;(param)</code> <code>make_shared&lt;T&gt;(param)</code>
Zugriff	<code>*p</code> <code>p-&gt;member</code>
ohne Zähler	<code>weak_ptr&lt;T&gt;(sharedptr)</code>
wieder zählen	<code>sp = w.lock()</code>

## Zufallszahlen <random>

Entropiequelle	<code>random_device</code>
Zufallsgenerator	<code>default_random_engine(rd)</code> <code>mt19937(rd)</code> <code>minstd_rand(rd)</code>
Verteilungen	<code>uniform_int_distribution(m, n)</code> <code>uniform_real_distribution(a, b)</code> <code>normal_distribution(<math>\mu, \sigma</math>)</code>
Zufallswert	<code>dist(gen)</code>

## Bitfolgen <bitset>

festen Größe $N$	<code>bitset&lt;N&gt;</code>
aus Zahl	<code>bitset(ulong)</code>
aus string	<code>bitset(s, pos, n, '0', '1')</code>
Bits setzen	<code>b.set()</code> <code>b.set(i)</code>
löschen	<code>b.reset()</code> <code>b.reset(i)</code>
negieren	<code>b.flip()</code> <code>b.flip(i)</code>
gesetzte Bits	<code>b.count()</code> <code>b.any()</code> <code>b.none()</code>
Konversion	<code>b.to_string()</code> <code>b.to_ulong()</code>

## Funktoren <functional>

Objektklassen mit überladenem operator()

einstellig	<code>unary_function&lt;Arg, Res&gt;</code>
$f(x) \mapsto -x$	<code>negate&lt;T&gt;</code>
$f(x) \mapsto !x$	<code>logical_not&lt;T&gt;</code>
zweistellig	<code>binary_function&lt;A<sub>1</sub>, A<sub>2</sub>, Res&gt;</code>
$f(x, y) \mapsto x+y$	<code>plus&lt;T&gt;</code>
$f(x, y) \mapsto x-y$	<code>minus&lt;T&gt;</code>
$f(x, y) \mapsto x*y$	<code>multiplies&lt;T&gt;</code>
$f(x, y) \mapsto x/y$	<code>divides&lt;T&gt;</code>
$f(x, y) \mapsto x\%y$	<code>modulus&lt;T&gt;</code>
$f(x, y) \mapsto x==y$	<code>equal_to&lt;T&gt;</code>
$f(x, y) \mapsto x!=y$	<code>not_equal_to&lt;T&gt;</code>
$f(x, y) \mapsto x>y$	<code>greater&lt;T&gt;</code>
$f(x, y) \mapsto x<y$	<code>less&lt;T&gt;</code>
$f(x, y) \mapsto x>=y$	<code>greater_equal&lt;T&gt;</code>
$f(x, y) \mapsto x<=y$	<code>less_equal&lt;T&gt;</code>
$f(x, y) \mapsto x\&\&y$	<code>logical_and&lt;T&gt;</code>
$f(x, y) \mapsto x\ \ y$	<code>logical_or&lt;T&gt;</code>

Negierer/Binder

$f(args) \mapsto !f(args)$	<code>not_fn(f)</code>
$f(args) \mapsto f(fewer)$	<code>bind(f, args)</code>

Methodenzeiger `mem_fn(Klasse::methode)`

Funktionszeiger `function<R(ParamTypen)>`

Beispiele:

```
int a[4] = { 1, 9, 6, 3 };
sort(a, a+4, greater{}); // 9 6 3 1
transform(a, a+4, a, negate{}); // -9 -6 -3 -1
function<int(int)> f = [](int x){ return -x; };
transform(a, a+4, a, f); // 9 6 3 1
```

## Iteratoren <iterator>

### Iteratorkategorien

Output	<i>mit Operatoren</i>	* ++
Input		== != * -> ++
Forward	<i>zusätzlich</i>	=
Bidirectional	<i>zusätzlich</i>	--
Random access	<i>zusätzlich</i>	< <= > >= + - += -= []
Reverse (Bidirectional)	<i>arbeitet auf mit vertauschter Richtung</i>	ri.base() ++ --
Iterator <i>n</i> Stellen weiterrücken		advance(it, n)
Abstand (Input-)Iteratoren		distance(f, l)

```
begin() ==> end()
|      ++   |
[.....)
^ |      ++ ^ |
rend() <== rbegin()
```

### Iterator-Adapter

Einfüger in Container <i>C</i>	
an Position	insert_iterator< <i>C</i> >
am Anfang	front_insert_iterator< <i>C</i> >
am Ende	back_insert_iterator< <i>C</i> >
Erzeuger-	inserter( <i>C</i> , <i>pos</i> )
Funktionen	front_inserter( <i>C</i> ) back_inserter( <i>C</i> )

*Beispiel:*

```
copy(first, last, back_inserter(c2));
```

Ausgabestrom-	
Iteratoren	ostream_iterator< <i>T</i> >
Konstruktor	o(strom, trenner_opt)

*Beispiel:*

```
ostream_iterator<int> o(cout, ", ");
*o = 123; // cout << "123, ";
o++;
```

Eingabestrom-	
Iteratoren	istream_iterator< <i>T</i> >
Konstruktor	i(strom_opt)

*Beispiel:*

```
istream_iterator<int> in(cin);
istream_iterator<int> end;
while (in != end)
{ wert = *in; // Wert liefern
  ++in;      // neuen Wert einlesen
}
```

## Zeichenketten <string\_view>

..._literals	"..."sv
ab Zeiger <i>p</i>	string_view( <i>p</i> )
<i>n</i> Zeichen	string_view( <i>p</i> , <i>n</i> )
Zuweisungen	s=s2
Vergleiche	< <= == >= > != compare(s2, pos2_opt, n2_opt) compare(pos, n, s2, pos2, n2)
Suchen liefert	npos bei Misserfolg
von vorn	s.find(params)
von hinten	s.rfind(params)
falls in Liste	s.find_first_of(params)
nicht in Liste	s.find_first_not_of(params)
letzte ...	s.find_last_of(params) s.find_last_not_of(params)
Iteratoren	s.begin() s.end()
lesend	s.cbegin() s.cend()
rückwärts	s.rbegin() s.rend() s.crbegin() s.crend()
Anzahl Zeichen	s.size()
leer	s.empty()
Teilstring	s.substr(i, n)
Kopiere ab <i>s</i> [ <i>i</i> ]	s.copy( <i>p</i> , <i>n</i> , <i>i</i> =0)
nach char-Feld <i>p</i>	max. <i>n</i> Zeichen, ohne '\0'
& <i>s</i> [0]	s.data() nicht terminiert!
<i>n</i> Zeichen	s.remove_prefix( <i>n</i> )
entfernen	s.remove_suffix( <i>n</i> )

## Zeichenketten <string>

string_literals	"..."s
zusätzlich zu	string_view:
Konstruktoren	mit <i>params</i>
<i>n</i> ab <i>s</i> [ <i>i</i> ]	string( <i>s</i> , <i>i</i> =0, <i>n</i> =npos)
aus char[]	string( <i>ptr</i> , <i>n</i> _opt)
<i>n</i> mal <i>c</i>	string( <i>n</i> , <i>c</i> )
Bereich	string( <i>first</i> , <i>last</i> ) string(string_view) to_string( <i>x</i> )
Verkettungen	s += s1+s2
Anhängen	s.append(params)
Einfügen bei	s.insert(ipos, params) s.insert(iter, params)
Löschen	s.erase(iter)
<i>n</i> ab <i>s</i> [ <i>i</i> ]	s.erase( <i>i</i> , <i>n</i> )
Bereich	s.erase(from, to)
Ersetzen ab	s.replace(ipos, n, params)
Bereich	s.replace(from, to, params)
Inhalt löschen	s.clear()
Konversion	stoi( <i>s</i> ) stol( <i>s</i> ) stof( <i>s</i> ) stod( <i>s</i> ) string_view(langlebiger_s)

## Zeichenarten <cctype>

Kleinbuchstabe	<code>tolower(c)</code>
Großbuchstabe	<code>toupper(c)</code>
klein?	<code>islower(c)</code>
groß?	<code>isupper(c)</code>
Buchstabe?	<code>isalpha(c)</code>
Buchstabe oder Ziffer?	<code>isalnum(c)</code>
Ziffer 0...9?	<code>isdigit(c)</code>
Hexziffer 0...9 A...F a...f?	<code>isxdigit(c)</code>
Leerraum, Tab, Zeilenende?	<code>isspace(c)</code>
Satzzeichen?	<code>ispunct(c)</code>
Steuerzeichen?	<code>iscntrl(c)</code>
druckbar?	auch ' ' <code>isprint(c)</code> ohne ' ' <code>isgraph(c)</code>

## Reguläre Ausdrücke <regex>

Raw string <i>s</i>	<code>R"delim(...)delim"</code>
Konstruktor	<code>regex(s, type_opt)</code>
Typ	<code>ECMAScript, basic, grep, ...</code>
Übereinstimmung	<code>regex_match(s, rex)</code>
Suchergebnis	<code>regex_match m</code>
Suche	<code>regex_search(s, m, rex)</code>
gesamt	<code>m[0]</code>
Teile	<code>m[1]...m[m.size()-1]</code>
Ersetzen	<code>regex_replace(s, rex, new)</code>

## Mathematik <cmath>

Betrag, runden	<code>fabs(x) round(x)</code>
$\lceil x \rceil$ $\lfloor x \rfloor$	<code>ceil(x) floor(x)</code>
$x^y$ $\sqrt{x}$	<code>pow(x, y) sqrt(x)</code>
Pythagoras	<code>hypot(x, y, z_opt)</code>
$e^x$ $\ln x$ $\lg x$	<code>exp(x) log(x) log10(x)</code>
trigonometrisch	<code>sin(x) cos(x) tan(x)</code>
Arcusfunktionen	<code>asin(x) acos(x) atan(x)</code>
im Kreis $\neq (0, 0)$	<code>atan2(y, x)</code>
Hyberbelfkt.	<code>sinh(x) cosh(x) tanh(x)</code>

## Komplexe Zahlen <complex>

Spezialisierungen	<code>complex&lt;T&gt;</code>
für <i>T</i> =	<code>float double long double</code>
Realteil	<code>c.real() real(c)</code>
Imaginärteil	<code>c.imag() imag(c)</code>
Betrag <i>r</i>	<code>abs(c)</code>
Winkel $\phi$	<code>arg(c)</code>
Betragsquadrat	<code>norm(c)</code>
Konjugierte	<code>conj(c)</code> <code>polar(r, phi)</code>
Funktionen aus	<cmath>

## Zahlen-Wertebereiche <limits>

Schablone <code>numeric_limits&lt;T&gt;</code>	
Angaben abrufbar	<code>is_specialized</code>
kleinster Wert	<code>min()</code>
größter Wert	<code>max()</code>
Anzahl Ziffern (Basissystem)	<code>digits</code>
im Dezimalsystem	<code>digits10</code>
vorzeichenbehaftet	<code>is_signed</code>
ganzzahlig	<code>is_integer</code>
beschränkt	<code>is_bounded</code>
exakt	<code>is_exact</code>
Überlauf möglich	<code>is_modulo</code>
Zahlenbasis	<code>radix</code>
IEC 559 Gleitkommatyp	<code>is_iec559</code>
kleinstes <i>e</i> mit <code>radix<sup>e</sup></code>	<code>min_exponent</code>
mit $10^e$	<code>min_exponent10</code>
größtes ...	<code>max_exponent</code> <code>max_exponent10</code>
Wert für $\infty$ verfügbar?	<code>has_infinity</code>
$\infty$	<code>infinity()</code>
kleinstes $\epsilon$ mit $1 + \epsilon > 1$	<code>epsilon()</code>
max. Rundungsfehler	<code>round_error()</code>
Rundungsart	<code>round_style</code>
<code>round_indeterminate</code>	
<code>round_toward_zero</code>	
<code>round_to_nearest</code>	
<code>round_toward_infinity</code>	
<code>round_toward_neg_infinity</code>	

## Dateisystem <filesystem>

in namespace `std::filesystem` (Auswahl)

Pfade	<code>path(s) current_path()</code>
vergleichen	<code>&lt; ... == !=</code>
verketteten	<code>p/=p2 p/p2</code>
zerlegen	<code>p.root_name()</code> <code>p.root_directory()</code> <code>p.root_path()</code> <code>p.relative_path()</code> <code>p.parent_path() p.filename()</code> <code>p.stem() p.extension()</code>
abfragen	<code>exists(p) is_directory(p)</code> <code>file_size(p)</code> <code>s. &lt;chrono&gt; last_write_time(p)</code>
Berechtigungen	<code>status(p).permissions()</code>
<code>perms::</code>	<code>none [owner group others]_</code> <code>[read write exec all] all mask</code> <code>sticky_bit set_uid set_gid</code>
anlegen, kopieren	<code>create_directories(p)</code> <code>copy(from, to)</code>
Pfade durchlaufen	<code>directory_iterator(p)</code>
(mit Unterverz.)	<code>recursive_...</code>
löschen	<code>remove(p) remove_all(p)</code>
Ausnahme	<code>filesystem_error</code>

## Ein-/Ausgabeströme <iostream>

Ausgabeströme (ostream)	cout cerr
Eingabeströme (istream)	cin
formatierte Ausgabe	<i>os</i> << <i>wert</i>
formatierte Eingabe	<i>is</i> >> <i>variable</i>
ein Zeichen <i>c</i> schreiben	<i>os.put(c)</i>
Vorausschau	<i>is.peek()</i>
ein Zeichen <i>c</i> lesen	<i>is.get(c)</i>
Zeichenkette <code>char s[n]</code> lesen	<i>is.getline(s,n)</i>
string <i>s</i> lesen	<code>getline(is,s)</code>
max. <i>n</i> char bis <i>c</i> übergehen	<i>is.ignore(n,c)</i>
bisher erfolgreich?	<code>if(os) ...</code>
solange Strom gültig	<code>while(is) ...</code>
Fehlerzustand zurücksetzen	<i>stream.clear()</i>

## Formatierung mit Manipulatoren <iomanip>

Eingabe	<i>is</i> >> <i>manip</i>
Ganzzahlbasis	dec hex oct
Leerraum (nicht) übergehen	ws noskipws
Ausgabe	<i>os</i> << <i>manip</i>
Zeilenvorschub	endl
Puffer leeren	flush
logische Werte	noboolalpha
Zahldarstellung	noshowpos noshowpoint
Nichtganzzahlen	fixed scientific
Genauigkeit	setprecision( <i>n</i> )
Ganzzahlbasis	dec hex oct noshowbase
Hexziffern, e/E	nouppercase
Ausgabebreite	setw( <i>n</i> )
Ausrichtung	left internal right
Füllzeichen	setfill( <i>c</i> )
Zeit ausgeben	put_time( <i>tptr</i> , "format")

```
cout<<fixed<<showpos<<setprecision(2)
  <<right<<setw(10)<<x<<endl;
```

## Stringströme <sstream>

Eingabestrom	istringstream <i>is(s)</i>
Ausgabestrom	ostringstream <i>os</i>
alle Ausgaben	<i>os.str()</i>
E-/A-Strom	stringstream <i>ss</i>

## I/O-Operatoren für Typ überladen

```
ostream& operator<<(ostream& os, Typ const& x)
{ // ...
  return os;
}
istream& operator>>(istream& is, Typ& x)
{ // ...
  return is;
}
```

## Dateiströme <fstream>

Eingabedatei	ifstream <i>is(name)</i>
Ausgabedatei	ofstream <i>os(name)</i>
E-/A-Datei	fstream <i>fs(name,modus)</i>
Modi aus ios	out ate in app binary
Beispiel:	ifstream d("a.txt", ios::in ios::binary)
unformatiert	<i>is.read(adresse,nbytes)</i>
lesen, schreiben	<i>os.write(adresse,nbytes)</i>
positionieren	<i>is.seekg(pos)</i>
relativ zu Position	<i>os.seekp(±n,ios::bezug)</i> <i>bezug = beg cur end</i>
Position erfragen	<i>is.tellg()</i> <i>os.tellp()</i>
Datei schließen	<i>fs.close()</i> (automatisch)

## Uhr und Zeit <chrono>, SI-Vorsätze <ratio>

Uhren	high_resolution_clock steady_clock system_clock
Zeitpunkt	<i>uhr::now()</i>
konvertieren	system_clock::to_time_t( <i>t</i> ) system_clock::from_time_t( <i>t</i> )
Zeitspannen	<i>t1.time_since_epoch()</i>
berechnen	<i>t2&gt;=t1 t2-t1 t1+dt</i>
Ticks	<i>dt.count()</i>
Zeiteinheiten	nanoseconds ... minutes hours
umrechnen	duration_cast<Ziel>(dt)
ms-Bruchteile	Ziel: duration<double, milli>
chrono_literals	2h+3min+4ms+5us
Vorsätze <ratio>	yocto zepto atto femto pico
10 <sup>-24</sup>  18 ... 10 <sup>+18</sup>  24	nano micro milli centi deci
je nach intmax_t	deca hecto kilo mega giga tera peta exa zetta yotta

## Zeitfunktionen <ctime>

Systemzeit time_t	time( <i>tptr</i> ) (Sek. seit 1970)
Zeitdifferenz	difftime( <i>t2,t1</i> )
lokale und	localtime( <i>tptr</i> )
Weltzeit → tm	gmtime( <i>tptr</i> )
struct tm	tm_sec tm_min tm_hour tm_mday tm_mon 0...11 tm_year ab 1900
Wochentag	tm_wday So=0...Sa=6
TagNr/Sommer	tm_yday tm_isdst
lokal → System	mktime( <i>tmptr</i> )
Zeichenkette	asctime( <i>tmptr</i> ) ctime( <i>time_t_ptr</i> )

Nur für Ausbildungszwecke. Recht auf Fehler vorbehalten.  
Unvollständig, inkonsistent, ... Hinweise willkommen.

(c) René Richter 2004–2020 namespace-cpp.de