

Kurzreferenz C++

Programmstruktur

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

Funktionen

anmelden	<code>Typ f(parameter);</code>
optimierbar	<code>inline ...</code>
festlegen	<code>Typ f(parameter)</code> <code>{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>{anweisungen};</code>
aufrufen	<code>f(argumente);</code> <code>auto ergebnis=f(argumente);</code>

Module

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

Präprozessoranweisungen

Einbinden	
Bibliothek	<code>#include <datei></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) < (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	<code>ausdruck;</code>
Anweisungsblock	<code>{anweisungen}</code>
Rückgabe aus Funktion	<code>return ergebnis;</code>
void-Funktion verlassen	<code>return;</code>
Sprung	
ans 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) anweisung1</code>
kann entfallen	<code>else opt anweisung2</code>
mehrfach	<code>switch(init; opt ausdruck) {</code>
Durchläufer	<code>case wert1: [[fallthrough]]</code>
jeder Fall	<code>case wert2: 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 <cassert></code>
Zusicherung	<code>assert(test);</code>
bei Übersetzung	<code>static_assert(test, meldung_opt);</code>
Ausnahme werfen	<code>throw ausdruck;</code>
weiterwerfen	<code>throw; (in catch-Block)</code>
möglich	<code>try {anweisungen}</code>
fangen	<code>catch(Typ& 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 / oktäl	<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	<code>'\101' '\xFF' u8'ä'</code>
Zeichenkette utf8	<code>"Hallo" u8"Welt"</code>

Typen

hergeleitet / ohne logisch, Zeichen ganzzahlig Modifizierer nichtganzzahlig Umbenennung	auto void bool char wchar_t short int long signed unsigned float double using <i>neuename</i> = <i>Typ</i> ; typedef <i>Typ neuename</i> ;
Aufzählung	enum class _{opt} <i>Typ</i> : <i>Basis</i> _{opt} { <i>name</i> = <i>wert</i> _{opt} ,...};
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 Folge beliebig	public/private/protected: <i>methoden, attribute</i> };
Zugriff erlaubt klassenbezogen Attribut	friend <i>funktion/klasse</i> static <i>methode/variable</i> <i>Typ name</i> = <i>wert</i> _{opt} ;
Methodenkopf Modifizierer polymorph	<i>Typ f</i> (<i>parameter</i>) const overload/final virtual <i>virtuelleMethode</i> virtual <i>abstrakteMethode</i> =0
Destruktor	virtual _{opt} ~ <i>Typ</i> ()=default;
Konstruktor	explicit _{opt} <i>Typ</i> (<i>parameter</i>)
Kopie, verschieben	<i>Typ</i> (const <i>Typ</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 abgeleitet von überschreiben ergänzen	class <i>Abgeleitet</i> : <i>art Basis</i> , ... { <i>zu ändernde methoden</i> <i>zusatzkomponenten</i> };
Vererbungsart bei Rhombus	public/protected/private 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 auch	<i>Typ objekt</i> { <i>wertliste</i> }; <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 spezialisiert	template<typename <i>T</i> ...> <i>Typ</i> <i>Typ</i> < <i>T</i> ...> <i>fkt</i> < <i>T</i> ...>(...)

Variablen

Variable (mit Anfangswert) Anfangswertliste bei Strukturen	<i>Typ name</i> = <i>wert</i> _{opt} = _{opt} { <i>wert</i> , ...}
aus Struktur/Feld	auto [<i>x,y,z</i>]=...
Feld (Reihe, array) mit <i>n</i> Werten mehrdimensional Zeichenkette	<i>Typ name</i> [<i>n</i>] ={ <i>wert</i> ₀ , ...} <i>name</i> [<i>m</i>][<i>n</i>]... 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	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 nach / vor Auswertung	++ <i>x</i> - - <i>x</i> ++
Vorzeichen	+ <i>x</i> - <i>x</i>
logisches / bitweises NICHT	! <i>x</i> ~ <i>x</i>
Zeigerinhalt, Adresse	* <i>ptr</i> & <i>objekt</i>
Speicher anfordern (Einzelobjekt) Feld	new <i>Typ</i> { <i>args</i> } _{opt} new <i>Typ</i> [<i>n</i>]
freigeben (Einzelobjekt/Feld)	delete [<i>opt ptr</i> <i>sizeof</i> (<i>name</i>) (<i>Typ</i>) <i>ausdruck</i>
Speicherbedarf in Byte Typecast	<i>objekt</i> .* <i>kptr</i> <i>ptr</i> ->* <i>kptr</i>
Komponentenauswahl über Objektzeiger	<i>x</i> * <i>y</i> <i>x</i> / <i>y</i> <i>m</i> % <i>n</i>
Punktrechnung, Divisionsrest	<i>x</i> + <i>y</i> <i>x</i> - <i>y</i>
Strichrechnung	<i>m</i> << <i>n</i> <i>m</i> >> <i>n</i>
Bits um <i>n</i> Stellen schieben	<i>x</i> << <i>y</i> <i>x</i> <= <i>y</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>
bitweises UND	<i>x</i> ^ <i>y</i>
bitweises Exklusiv-ODER	<i>x</i> <i>y</i>
bitweises ODER	<i>x</i> &&& <i>y</i>
logisches UND	<i>x</i> <i>y</i>
logisches ODER	<i>bed</i> ? <i>dann</i> : <i>sonst</i>
bedingter Ausdruck	throw <i>ausdruck</i>
Ausnahme werfen	<i>x</i> = <i>wert</i>
Zuweisung und Kurzschrift	<i>x</i> += <i>y</i> für <i>x</i> = <i>x</i> + <i>y</i>
für + - * / % << >> & ^	Liste von Ausdrücken ,

einstellige Operatoren und Zuweisungen von rechts,
alle anderen von links bindend.

Standard-Bibliothek (Auswahl)

in namespace `std`

Container

Allgemeine Container-Eigenschaften

Kopie	$C(C_2)$
aus Bereich	$C(f, l)$
Zuweisung	$C=C_2$
Tausch	$C.swap(C_2)$
Vergleich	$== \quad !=$
lexikographisch	$< \quad <= \quad >= \quad >$
ist leer?	$C.empty()$
Anzahl Werte	$C.size()$
max. Anzahl	$C.max_size()$
Iteratoren	$C.begin() \quad C.end()$
lesend	$C.cbegin() \quad C.cend()$
rückwärts	$C.rbegin() \quad C.rend()$
	$C.crbegin() \quad C.crend()$
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<T></code>
	<code>unordered_</code>	<code>multi</code>	<code>map<Key, Value></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<T < ></code>	
ungeordnet	<code>unordered_</code>	<code>multi</code>	<code>set<T ></code>
Kriterium < >	<code>less<T></code>	<code>hash<T></code>	

Assoziative Felder <map> <unordered_map>

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

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

Sequentielle Container

<code>vector<T></code>	<code>deque<T></code>	<code>list<T></code>	<code>forward_list<T></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	$C.pop_back()$		

dynamisches Feld <vector>

sequentuell	<code>vector<T></code>
Feldzugriff	$C[index] \quad C.at(index)$

doppelendige Schlange <deque>

wie <code>vector<T></code>	<code>deque<T></code>
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	$C.merge(list < >)$
Sortieren	$C.sort(< >)$
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<T></code>	$C.before_begin() \quad \dots_end()$ $C.splice_after(pos, list\dots)$

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, <)</code> <code>lower_bound(f, l, wert, <)</code> <code>upper_bound(f, l, wert, <)</code>
Bereich	<code>equal_range(f, l, wert, <)</code>
Minimum	<code>min(a, b, <)</code>
Maximum im Bereich (Position)	<code>max(a, b, <)</code> <code>min_element(f, l, <)</code> <code>max_element(f, l, <)</code>
Eingrenzen	<code>clamp(x, lo, hi, <)</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, <)</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 mit Funktor	<code>fill(f, l, wert)</code> <code>..._n(f, n, wert)</code> <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, F_2)</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, <)</code> <code>prev_permutation(f, l, <)</code>
Sortieren	<code>sort(f, l, <)</code> <code>stable_sort(f, l, <)</code> Teilbereich <code>partial_sort(f, mitte, l, <)</code> <code>..._copy(f, mitte, l, <)</code> bis zum n -ten <code>nth_element(f, nth, l, <)</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, <)</code> <code>inplace_... (f, mitte, l, <)</code>
$[f, l] \supseteq [f_2, l_2)?$	<code>includes(f, l, f_2, l_2, <)</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, <)</code>

numerische Algorithmen <numeric>

ggT / kgV	<code>gcd(m, n)</code> <code>lcm(m, n)</code>
Summe	<code>accumulate(f, l, init, ⊕)</code>
Teilsummen	<code>partial_sum(f, l, to, ⊕)</code>
Nachbardifferenz	<code>adjacent_difference(f, l, to, ⊖)</code>
Skalarprodukt	<code>inner_product(f, l, f_2, init, ⊕, ⊙)</code>

Zufallszahlen <random>

Entropiequelle	<code>random_device</code>
Zufallsgenerator	<code>mt19937(rd)</code> <code>minstd_rand(rd)</code>
Verteilungen $[m, n]$ $[a, b)$ $N(\mu, \sigma^2)$	<code>uniform_int_distribution(m, n)</code> <code>uniform_real_distribution(a, b)</code> <code>normal_distribution(μ, σ)</code>
Zufallswert	<code>dist(gen)</code>

Legende:

Iterator-Bereiche $[first, last)$	f, l f_2, l_2
Iterator Anfang Zielbereich	to
verallgemeinerte Funktionen	F, F_2
Generator $x = Gen()$	Gen
Prädikat $bool P(x)$	P
zweistellig $bool P_2(x, y)$	P_2
Vergleich $bool <(x, y)$	$x < y$
Binäroperator	$x \oplus y$ $x \odot y$
optionales Argument, z.B. $<$	$<$

Zubehör

Container-Adapter <stack> <queue>

ist leer?	<code>a.empty()</code>
Anzahl Elemente	<code>a.size()</code>
Vergleiche	<code>< == ...</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 \triangleleft	<code>less<T></code>
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, <>)</code>
* $(l-1)$ dazu	<code>push_heap(f, l, <>)</code>
* f nach hinten	<code>pop_heap(f, l, <>)</code>
Heap-Sort	<code>sort_heap(f, l, <>)</code>

Array <array> <valarray>

festen Größe	<code>array<T, N></code>
Vergleiche	<code>< == ...</code>
dynamische Größe	<code>valarray<T></code>
Elementgruppen	<code>v[slice(pos, n, dist)]</code>
für $0 \leq i < n$	<code>v[pos+i*dist]</code>
Operatoren	<code>+ - * / % & ^ << >> && =</code>
Funktionen aus	<cmath>

Bitfolgen <bitset>

festen Größe N	<code>bitset<N></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>

Wrapper

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

Funktoren <functional>

Objektclassen mit überladenen operator()

einstellig	<code>unary_function<Arg, Res></code>
$f(x) \mapsto -x$	<code>negate<T></code>
$f(x) \mapsto !x$	<code>logical_not<T></code>
zweistellig	<code>binary_function<A₁, A₂, Res></code>
$f(x, y) \mapsto x+y$	<code>plus<T></code>
$f(x, y) \mapsto x-y$	<code>minus<T></code>
$f(x, y) \mapsto x*y$	<code>multiplies<T></code>
$f(x, y) \mapsto x/y$	<code>divides<T></code>
$f(x, y) \mapsto x\%y$	<code>modulus<T></code>
$f(x, y) \mapsto x==y$	<code>equal_to<T></code>
$f(x, y) \mapsto x!=y$	<code>not_equal_to<T></code>
$f(x, y) \mapsto x>y$	<code>greater<T></code>
$f(x, y) \mapsto x<y$	<code>less<T></code>
$f(x, y) \mapsto x>=y$	<code>greater_equal<T></code>
$f(x, y) \mapsto x<=y$	<code>less_equal<T></code>
$f(x, y) \mapsto x\&\&y$	<code>logical_and<T></code>
$f(x, y) \mapsto x\ \ y$	<code>logical_or<T></code>
Negierer/Binder	
$f(args) \mapsto !f(args)$	<code>not_fn(args)</code>
$f(args) \mapsto f(fewer)$	<code>bind(f, args)</code>
Methodenzeiger	<code>mem_fn(Klasse::methode)</code>
Funktionszeiger	<code>function<R(ParamTypen)></code>

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</i>	<i>ri.base()</i>
<i>mit vertauschter Richtung</i>		++ --
Iterator <i>n</i> Stellen weiterrücken		<i>advance(it, n)</i>
Abstand (Input-)Iteratoren		<i>distance(f, l)</i>

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

Iterator-Adapter

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

Beispiel:

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

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

Beispiel:

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

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

Beispiel:

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

Zeichenketten <string_view>

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

Zeichenketten <string>

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

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 s	<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>
Hyperbelfkt.	<code>sinh(x) cosh(x) tanh(x)</code>

Komplexe Zahlen <complex>

Spezialisierungen	<code>complex<float></code> <code>complex<double></code> <code>complex<long double></code>
Realteil	<code>c.real() real(c)</code>
Imaginärteil	<code>c.imag() imag(c)</code>
Betrag r	<code>abs(c)</code>
Winkel ϕ	<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<T></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 e mit <code>radix^e</code>	<code>min_exponent</code>
mit 10^e	<code>min_exponent10</code>
größtes ...	<code>max_exponent</code> <code>max_exponent10</code>

Wert für ∞ verfügbar?	<code>has_infinity</code>
∞	<code>infinity()</code>
kleinstes ϵ 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>< ... == !=</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>last_write_time(p)</code>
s. <chrono>	<code>status(p).permissions()</code>
Berechtigungen	<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 <i>char s[n]</i> lesen	<i>is.getline(s,n)</i>
<i>string s</i> lesen	<i>getline(is,s)</i>
max. <i>n char</i> bis <i>c</i> übergehen	<i>is.ignore(n,c)</i>
bisher erfolgreich?	<i>if(os) ...</i>
solange Strom gültig	<i>while(is) ...</i>
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>tprtr</i> , " <i>format</i> ")

```
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, const Typ& 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
<i>Beispiel:</i>	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	<i>os.seekp(±n,ios::bezug)</i>
Position	<i>bezug = beg cur end</i>
Position	<i>is.tellg()</i>
erfragen	<i>os.tellp()</i>
Datei schließen	<i>fs.close()</i> (<i>automatisch</i>)

Zeitfunktionen <ctime>

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

Uhren und Zeitspannen <chrono>

Uhren	high_resolution_clock steady_clock system_clock
Zeitpunkt	uhr::now()
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>=t1 t2-t1 t1+dt</i>
Ticks	<i>dt.count()</i>
Zeiteinheiten	duration< <i>rep, ratio</i> > nanoseconds ... minutes hours
umrechnen	duration_cast< <i>Ziel</i> >(dt)
<i>gebrochen</i>	duration<double, milli>
<ratio>	yocto zepto atto femto pico
SI-Vorsätze	nano micro milli centi deci
10 ^{-18/24} ...10 ^{+18/24}	deca hecto kilo mega giga tera
<i>je nach intmax_t</i>	peta exa zetta yotta
chrono_literals	2h+3min+4ms+5us

Nur für Ausbildungszwecke.
Hinweise willkommen.
Recht auf Fehler vorbehalten.