

C++ Kurzreferenz

Programmstruktur

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

Funktionen

anmelden	<code>typ f(parameter);</code>
festlegen	<code>typ f(parameter) {<i>anweisungen</i>}</code>
optimierbar	<code>inline typ f(parameter) {<i>anweisungen</i>}</code>
Lambda-Ausdruck	<code>auto f = [<i>capturelist</i>](parameter) {<i>anweisungen</i>};</code>
Parameterliste	<code>typ name, ...</code>
Vorgabewert	<code>typ name=wert</code>
aufrufen	<code>f(argumente);</code>

Module

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

Abschluss Anweisung	<code>ausdruck;</code>
Anweisungsblock	<code>{<i>anweisungen</i>}</code>
Rücksprung aus Funktion	<code>return wert;</code>
aus void-Funktion	<code>return;</code>
Sprung	
aus Schleife / switch	<code>break;</code>
ans Schleifenblock-Ende	<code>continue;</code>
innerhalb einer Funktion	<code>goto marke;</code>
Sprungziel	<code>marke:</code>

Entscheidungen

einfach	<code>if(<i>init</i>; <i>opt</i> <i>bedingung</i>) <i>anw1</i></code>
optional	<code>else <i>anw2</i></code>
mehrfach	<code>switch(<i>init</i>; <i>opt</i> <i>ausdruck</i>) {</code>
Durchläufer	<code>case <i>wert1</i>: [[<i>fallthrough</i>]]</code>
jeder Fall	<code>case <i>wert2</i>: <i>anweisungen</i></code>
mit Abschluss	<code>break;</code>
sonst-Zweig	<code>default: <i>anweisungen</i></code>
	<code>}</code>

Wiederholungen

kopfgesteuert	<code>while(<i>bedingung</i>)</code>
	<code><i>anweisung</i></code>
fußgesteuert	<code>do <i>anweisung</i></code>
	<code>while(<i>bedingung</i>);</code>
Zählschleife	<code>for(<i>start</i>; <i>bedingung</i>; <i>schritt</i>)</code>
	<code><i>anweisung</i></code>
über Bereich	<code>for(auto <i>wert</i>: {1,2,3})...</code>

Zusicherungen / Ausnahmebehandlung

bei Übersetzung	<code>static_assert(<i>bedingung</i></code>
sonst	<code>, <i>Meldung</i> <i>opt</i>);</code>
Ausnahme	
möglich	<code>try {<i>anweisungen</i>}</code>
fangen	<code>catch(<i>typ</i> <i>ausnahme</i>)</code>
behandeln	<code>{<i>anweisungen</i>}</code>
alle Typen	<code>catch(...){<i>anweisungen</i>}</code>
werfen	<code>throw <i>ausdruck</i>;</code>
weiterwerfen	<code>throw; (in catch-Block)</code>

Konstanten (Literele)

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>
oktal/hex/utf8	<code>'\101' '\xFF' u8'ä'</code>
Zeile, Tab, ...	<code>\n \t \r \\' \'\"</code>
Zeichenkette	<code>"Hallo" u8"Welt"</code>

Variablen

Variable	<i>typ name</i>
mit Anfangswert	=wert
Referenz	<i>typ& ref=alias</i>
Zeiger	<i>typ* ptr=adresse</i>
Feld (Reihe, array)	<i>typ name [n]</i>
mit n Werten	= <i>{wert₀, ...}</i>
mehrdimensional	<i>name [m] [n] ...</i>
Zeichenkette	<i>char s []="az"</i>
Aggregatvariable	<i>struct_{opt} typ name</i>
mit Attributwerten	= <i>{wert, ...}</i>
nur lesender Zugriff	<i>const ...</i>
trotzdem änderbar	<i>mutable ...</i>
beim Übersetzen berechenbar	<i>constexpr ...</i>
statisch / lokale Bindung	<i>static ...</i>
flüchtig, nicht optimieren	<i>volatile ...</i>
strukturierte Bindung	<i>auto [x,y,z]=...</i>

Operatoren (nach Rang geordnet)

Namensbereich-Auflösung	<i>bereich::name</i>
Funktionsaufruf	<i>funktion()</i>
Feldzugriff	<i>feld [index]</i>
Struktur-Komponente	<i>objekt.teil</i>
Zugriff über Zeiger	<i>ptr->teil</i>
Erhöhen, Absenken nach / vor Auswertung	<i>x++ x--</i> <i>++x --x</i>
Vorzeichen	<i>+x -x</i>
logisches / bitweises NICHT	<i>!x ~x</i>
Zeigerinhalt, Adresse	<i>*zeiger &objekt</i>
Speicher anfordern	<i>new typ [n]_{opt}</i>
freigeben (einzeln / Feld)	<i>delete_{opt} ptr</i>
Speicherbedarf in Byte	<i>sizeof (name)</i>
Typecast	<i>(typ) ausdruck</i>
Komponentenauswahl über Objektzeiger	<i>objekt.*kptr</i> <i>ptr->*kptr</i>
mal, durch, Divisionsrest	<i>x*y x/y m%n</i>
addieren, subtrahieren	<i>x+y x-y</i>
Bits um n schieben	<i>m<<n m>>n</i>
kleiner (oder gleich)	<i>x<y x<=y</i>
größer (oder gleich)	<i>x>y x>=y</i>
gleich / ungleich	<i>x==y x!=y</i>
bitweises UND	<i>x&y</i>
bitweises Exklusiv-ODER	<i>x^y</i>
bitweises ODER	<i>x y</i>
logisches UND	<i>x&&y</i>
logisches ODER	<i>x y</i>
bedingter Ausdruck	<i>bed?dann:sonst</i>
Zuweisung und Kurzschrift	<i>x=wert</i>
für + - * / % << >> & ^	<i>x+=y für x=x+y</i>
Ausnahme auslösen	<i>throw ausdruck</i>
Liste von Ausdrücken	,

Einstellige Operatoren, Zuweisungen von rechts, alle anderen von links bindend.

Typen

hergeleitet	<i>auto</i>
kein Typ	<i>void</i>
logisch, Zeichen	<i>bool char wchar_t</i>
ganzzahlig	<i>short int long</i>
Modifizierer	<i>signed unsigned</i>
nichtganzzahlig	<i>float double</i>
Umbenennung	<i>typedef Typ neuername;</i> <i>using neuername=Typ;</i>
Aufzählung	<i>enum class_{opt} Typ</i> <i>{name=wert_{opt},...};</i>
Überlagerung	<i>union Typ {komponenten};</i>

Klassen, Strukturen

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

Schablonen

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

Standard-Bibliothek (Auswahl)

Container

Allgemeine Container-Eigenschaften

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	$C.begin() \quad C.end()$
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><</code>	<code>H</code>	<code>less<T> / 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><</code>	<code>H</code>	<code>less<K> / 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>

sequentiell	<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>	$C.before_begin() / end()$
	$C.splice_after(pos, list...)$

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 Bereich	<code>binary_search(f, l, wert, < >)</code> <code>lower_bound(f, l, wert, < >)</code> <code>upper_bound(f, l, wert, < >)</code> <code>equal_range(f, l, wert, < >)</code>
Minimum Maximum im Bereich (Position)	<code>min(a, b, < >)</code> <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 Sortierfolge $[f, l) < [f_2, l_2)$ Unterschied ab	<code>equal(f, l, f_2, l_2, P_2)</code> <code>lexicographical_compare(f, l, f_2, l_2, < >)</code> <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 Teilbereich bis zum n-ten	<code>sort(f, l, < >)</code> <code>stable_sort(f, l, < >)</code> <code>partial_sort(f, mitte, l, < >)</code> <code>..._copy(f, mitte, l, < >)</code> <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>
$[f, l) \supseteq [f_2, l_2)?$ $[f, l) \cup [f_2, l_2)$ $[f, l) \cap [f_2, l_2)$ $[f, l) \setminus [f_2, l_2)$ $[f, l) \Delta [f_2, l_2)$	<code>includes(f, l, f_2, l_2, < >)</code> <code>set_union...</code> <code>set_intersection...</code> <code>set_difference...</code> <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>

Stapel <code>stack<T></code>	
ist leer?	<code>s.empty()</code>
Anzahl Elemente	<code>s.size()</code>
Vergleiche	<code>< == ...</code>
Einfügen Wert x	<code>s.push(x)</code>
Entfernen (ohne Rückgabe)	<code>s.pop()</code>
oberstes Element	<code>s.top()</code>

Warteschlange `queue<T>`

ist leer?	<code>q.empty()</code>
Anzahl Elemente	<code>q.size()</code>
Vergleiche	<code>< == ...</code>
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>

... mit Vordrängeln `priority_queue<T, C, <>`

Sortierkriterium $<$	<code>less<T></code>
ist leer?	<code>p.empty()</code>
Anzahl Elemente	<code>p.size()</code>
Einfügen Wert x	<code>p.push(x)</code>
Entfernen (ohne Rückgabe)	<code>p.pop()</code>
oberstes Element	<code>p.top()</code>

<i>Heapfunktionen</i>	<i>aus <algorithm></i>
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>

feste Größe	<code>array<T, N></code>
dyn. Größe	<code>valarray<T></code>
Elementauswahl	<code>v[slice(pos, n, dist)]</code>
für $0 \leq i < n$	<code>v[pos+i*dist]</code>
Operatoren	<code>+ - * / % & ^ << >></code>
	<code>+ && < <= >= > == !=</code>
<cmath>	<code>sqrt(v) ...</code>

Bitfolgen <bitset>

feste Größe N	<code>bitset<N></code>
aus Zahl	<code>bitset(ulong)</code>
Zeichenkette	<code>bitset(str, pos, n)</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>

Objektklassen mit überladenem 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

	<i>Operatoren</i>
Output	* ++
Input	== != * -> ++
Forward	<i>zusätzlich</i> =
Bidirectional	<i>zusätzlich</i> --
Random Access	<i>zusätzlich</i> < <= > >= + - += -= []
Reverse (Bidirectional) <i>vertauschte Wirkung</i> <i>versetzt darunterliegend</i>	++ -- <i>ri.base()</i>
<pre>begin() --> end() v ++ v [.....) ^ ++ ^ rend() <-- rbegin()</pre>	
Iterator <i>in</i> um <i>n</i> weitersetzen	<i>advance(in, n)</i>
Abstand Input-Iteratoren	<i>distance(f, l)</i>

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-Funktionen	<i>inserter(C, pos)</i> <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, trennstring)</i>

Beispiel:

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

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

Beispiel:

```
istream_iterator<int> end; // ohne Strom
istream_iterator<int> in{cin};
// liest und puffert 1. Wert
while(in != end) {
    wert = *in; // liefern
    ++in;      // neuen Wert einlesen
}
```

Zeichenketten <string_view>

Literal	"..."sv
ab Zeiger <i>p</i> <i>n</i> Zeichen	<i>string_view(p)</i> <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()</i> <i>s.end()</i>
lesend	<i>s.cbegin()</i> <i>s.cend()</i>
rückwärts	<i>s.rbegin()</i> <i>s.rend()</i> <i>s.crbegin()</i> <i>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> nach char-Feld <i>p</i> & <i>s[0]</i>	<i>s.copy(p, n, i=0)</i> <i>max. n Zeichen, ohne '\0'</i> <i>s.data()</i>
<i>n</i> Zeichen entfernen	<i>s.remove_prefix(n)</i> <i>s.remove_suffix(n)</i>

Zeichenketten <string>

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

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 \((0,0)	<code>atan2(y, x)</code>
Hyberbelfkt.	<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>	
Spezialisierungen für alle numerischen Typen	
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 Fließkommatyp	<code>is_iec559</code>
kleinstes e mit radix^e	<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>...toward_neg_infinity</code>
und weitere ...	

Beispiel:

```
cout << numeric_limits<long>::min() << ' ' << numeric_limits<long>::max();
```

Fehlererkennung <cassert>

Prüfung	<code>assert(Bedingung)</code>
bei Misserfolg	<i>Ausschrift, Programmende</i>
Test abschalten	<code>#define NDEBUG</code>
	vor <code>#include <cassert></code>

Hilfsfunktionen <cstdlib>

Kommando	<code>system(befehl)</code>
Programmende	<code>exit(fehlernum)</code>

Ein-/Ausgabeströme <iostream>

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

Formatierung mit Manipulatoren <iomanip>

Eingabe	is>>manip
Ganzzahlbasis	dec hex oct
Übergehen von	ws noskipws
Whitespaces	' ' '\t' '\n'
Ausgabe	os<<manip
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>) (flüchtig)
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</i> (<i>string</i>)
Ausgabestrom	ostringstream <i>os</i>
alle Ausgaben	os.str()
E-/A-Strom	stringstream <i>ss</i>

I/O-Operatoren für Typ T überladen

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

Dateiströme <fstream>

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

Zeitfunktionen <ctime>

Systemzeit	time(<i>tptr</i>) (Sek. ab 1970)
Zeitdifferenz	difftime(<i>t₂</i> , <i>t₁</i>)
lokale und	localtime(<i>tptr</i>)
Weltzeit	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>tmprtr</i>)
Zeichenkette	asctime(<i>tmprtr</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</i> .time_since_epoch()
berechnen	<i>t2</i> >= <i>t1</i> <i>t2</i> - <i>t1</i> <i>t1</i> + <i>dt</i>
Ticks	<i>dt</i> .count()
Zeiteinheiten	duration< <i>rep</i> , <i>ratio</i> > nanoseconds ... hours
umrechnen	duration_cast< <i>Ziel</i> >(<i>dt</i>)
gebrochen	duration<double, milli>
SI-Vorsatz	yocto zepto atto femto pico
<ratio>	nano micro milli centi deci
10 ⁻²⁴ ...10 ⁺²⁴ (abh. von <i>intmax_t</i>)	deca hecto kilo mega giga tera peta exa zetta yotta

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