Kurzreferenz C++

Programmstruktur

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

Funktionen

anmelden	*Typ f*`(`*parameter*`);`
optimierbar	`inline` ...
festlegen	*Typ f*`(`*parameter*`)`
	`{`*anweisungen*`}`
Parameterliste	*Typ name, ...*
Vorgabewert	*Typ name=wert*
Funktion aus	`auto` *f*`=`
Lambda-Ausdruck	`[`*einschlussliste*`](`*parameter*`)`
	`{`*anweisungen*`};`
aufrufen	*f*`(`*argumente*`);`
	`auto` *ergebnis=f*`(`*argumente*`);`

Module

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

Präprozessoranweisungen

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

Ablaufsteuerung

Anweisung	*ausdruck*`;`
Anweisungsblock	`{`*anweisungen*`}`
Rückgabe aus Funktion	`return` *ergebnis*`;`
`void`-Funktion verlassen	`return;`
Sprung	
ans Schleifenblock-Ende	`continue;`
aus Schleife / `switch`	`break;`
innerhalb einer Funktion	`goto` *marke*`;`
Sprungziel	*marke*`:`

Wiederholungen (Schleifen)

über Wertfolge	`for(`*Typ element*`:`*liste*`)` *anweisung*
Zählschleife	`for(`*init*`;`*bedingung*`;`*schritt*`)`
	anweisung
kopfgesteuert	`while(`*bedingung*`)` *anweisung*
fußgesteuert	`do` *anweisung*
	`while(`*bedingung*`);`

Entscheidungen

einfach	`if(`*init*$;_{opt}$ *bedingung*`)` *anweisung*$_1$
kann entfallen	`else`$_{opt}$ *anweisung*$_2$
mehrfach	`switch(`*init*$;_{opt}$ *ausdruck*`) {`
Durchläufer	`case` *wert*$_1$`: [[fallthrough]]`
jeder Fall	`case` *wert*$_2$`:` *anweisungen*
mit Abschluss	`break;`
sonst-Zweig	`default:` *anweisungen*
	`}`

Zusicherungen, Ausnahmebehandlung

Abschalten der	`#define NDEBUG`
Laufzeittests	`#include <cassert>`
Zusicherung	`assert(`*test*`);`
bei Übersetzung	`static_assert(`*test,meldung*$_{opt}$`);`
Ausnahme werfen	`throw` *ausdruck*`;`
weiterwerfen	`throw;` *(in `catch`-Block)*
möglich	`try {`*anweisungen*`}`
fangen	`catch(`*Typ*`&` *ausnahme*`)`
behandeln	`{`*anweisungen*`}`
alle anderen	`catch(...){`*anweisungen*`}`

Konstanten (Literale)

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

Typen

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

Klassen, Strukturen

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

Schablonen

Funktion	`template<typename` *T ...*`>` *fkt*
Struktur, Klasse	`template<typename` *T ...*`>` *Typ*
spezialisiert	*Typ*`<`*T ...*`>` *fkt*`<`*T ...*`>(...)`

Variablen

Variable (mit Anfangswert)	*Typ name*`=`*wert*$_{opt}$
Anfangswertliste bei Strukturen	`=`$_{opt}$`{`*wert, ...*`}`
aus Struktur/Feld	`auto[`*x,y,z*`]=...`
Feld (Reihe, array)	*Typ name*`[`*n*`]`
mit n Werten	`={`*wert*$_0$*, ...*`}`
mehrdimensional	*name*`[`*m*`][`*n*`]...`
Zeichenkette	`char s[]="az"`
Referenz	*Typ*`&` *ref*`=`*alias*
Zeiger	*Typ*`*` *ptr*`=`*adresse*
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	*bereich*`::`*name*
Funktionsaufruf	*funktion*`()`
Feldzugriff	*feld*`[`*index*`]`
Struktur-Komponente	*objekt*`.`*teil*
Zugriff über Zeiger	*ptr*`->`*teil*
Erhöhen, Absenken nach /	*x*`++` *x*`-`
vor Auswertung	`++`*x* `-`*x*
Vorzeichen	`+`*x* `-`*x*
logisches / bitweises NICHT	`!`*x* `~`*x*
Zeigerinhalt, Adresse	`*`*ptr* `&`*objekt*
Speicher anfordern (Einzelobjekt)	`new` *Typ*`{`*args*`}`$_{opt}$
Feld	`new` *Typ*`[`*n*`]`
freigeben (Einzelobjekt/Feld)	`delete[]`$_{opt}$ *ptr*
Speicherbedarf in Byte	`sizeof(`*name*`)`
Typecast	`(`*Typ*`)` *ausdruck*
Komponentenauswahl	*objekt*`.*`*kptr*
über Objektzeiger	*ptr*`->*`*kptr*
Punktrechnung, Divisionsrest	*x*`*`*y* *x*`/`*y* *m*`%`*n*
Strichrechnung	*x*`+`*y* *x*`-`*y*
Bits um *n* Stellen schieben	*m*`<<`*n* *m*`>>`*n*
kleiner (oder gleich)	*x*`<`*y* *x*`<=`*y*
größer (oder gleich)	*x*`>`*y* *x*`>=`*y*
gleich / ungleich	*x*`==`*y* *x*`!=`*y*
bitweises UND	*x*`&`*y*
bitweises Exklusiv-ODER	*x*`^`*y*
bitweises ODER	*x*`|`*y*
logisches UND	*x*`&&`*y*
logisches ODER	*x*`||`*y*
bedingter Ausdruck	*bed*`?`*dann*`:`*sonst*
Ausnahme werfen	`throw` *ausdruck*
Zuweisung und Kurzschrift	*x*`=`*wert*
für `+ - * / % << >> & | ^`	*x*`+=`*y* für *x*`=`*x*`+`*y*
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	`== !=`
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

[unordered_] [multi]	set<T>
[unordered_] [multi]	map<Key,Value>
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	[multi] set<T [◁]>
ungeordnet	unordered_ [multi] set<T [H]>
Kriterium [◁] [H]	less<T> hash<T>

Assoziative Felder `<map> <unordered_map>`

Schlüssel K,	[multi] map<K,V [◁]>
Wert V	unordered_ [multi] map<K,V [H]>
Eintrag	pair<const K,V>
Kriterium [◁] [H]	less<K> hash<K>
Wert-Zugriff	C`.at(`k`)` C`[`k`]` C`[`k`]=`v

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

Sequentielle Container

`vector<T> deque<T> list<T> forward_list<T>`	
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	vector<T>
Feldzugriff	C`[`*index*`]` C`.at(`*index*`)`

doppelendige Schlange `<deque>`

wie vector<T>	deque<T>
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]`)`
forward_list<T>	C`.before_begin()` *..._end()*
	C`.splice_after(`*pos*`,`*list...*`)`

Algorithmen `<algorithm>`

nicht modifizierend

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

modifizierend (wertändernd)

Tauschen	`swap(`a`,`b`)`
Kopieren	`copy(`f`,`l`,`to`)`
	`..._backward(`f`,`l`,`to`)`
Ausfüllen	`fill(`f`,`l`,`*wert*`)`
	`..._n(`f`,`n`,`*wert*`)`
mit Funktor	`generate(`f`,`l`,`*Gen*`)`
	`..._n(`f`,`n`,`*Gen*`)`
Ersetzen	`replace(`f`,`l`,`*alt*`,`*neu*`)`
	`..._if(`f`,`l`,`P`,`*neu*`)`
	`..._copy(`f`,`l`,`to`,`*alt*`,`*neu*`)`
	`..._copy_if(`f`,`l`,`to`,`P`,`*neu*`)`
Entfernen	`remove(`f`,`l`,`*wert*`)`
	`..._if(`f`,`l`,`P`)`
	`..._copy(`f`,`l`,`to`,`*wert*`)`
	`..._copy_if(`f`,`l`,`to`,`P`)`
ohne Dubletten	`unique(`f`,`l` [`P_2`])`
	`..._copy(`f`,`l`,`to` [`P_2`])`
Umrechnen	`transform(`f`,`l`,`to`,`F`)`
	`transform(`f`,`l`,`f_2`,`l_2`,`to`,`F_2`)`

mutierend (Reihenfolge ändernd)

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

numerische Algorithmen `<numeric>`

ggT / kgV	`gcd(`m`,`n`)` `lcm(`m`,`n`)`
Summe	`accumulate(`f`,`l`,`*init* [$\oplus$]`)`
Teilsummen	`partial_sum(`f`,`l`,`to` [$\oplus$]`)`
Nachbardifferenz	`adjacent_difference(`f`,`l`,`to` [$\ominus$]`)`
Skalarprodukt	`inner_product(`f`,`l`,`f_2`,`*init* [$\oplus$,$\odot$]`)`

Zufallszahlen `<random>`

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

Legende:	
Iterator-Bereiche [first,last)	$f,l \quad f_2,l_2$
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` $\triangleleft$`(x,y)`	x$\triangleleft$y
Binäroperator	x$\oplus$y x$\odot$y
optionales Argument, z.B. $\triangleleft$	[$\triangleleft$]

Zubehör

Container-Adapter `<stack> <queue>`

ist leer?	`a.empty()`
Anzahl Elemente	`a.size()`
Vergleiche	`< == ...`

Stapel `stack<T,`C_{opt}`>`

Einfügen Wert x	`st.push(`x`)`
Entfernen (ohne Rückgabe)	`st.pop()`
oberstes Element	`st.top()`

Warteschlange `queue<T,`C_{opt}`>`

Einfügen Wert x	`q.push(`x`)`
Entfernen (ohne Rückgabe)	`q.pop()`
erstes Element	`q.front()`
letztes Element	`q.back()`

... zum Vordrängeln `priority_queue<T,`C_{opt} `◁>`

Sortierkriterium ◁	`less<T>`
Einfügen Wert x	`pq.push(`x`)`
Entfernen (ohne Rückgabe)	`pq.pop()`
oberstes Element	`pq.top()`

mit Funktionen aus `<algorithm>`

Herstellen Heap	`make_heap(`f`,`l`◁)`
`*(`l`-1)` dazu	`push_heap(`f`,`l`◁)`
`*`f nach hinten	`pop_heap(`f`,`l`◁)`
Heap-Sort	`sort_heap(`f`,`l`◁)`

Array `<array> <valarray>`

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

Bitfolgen `<bitset>`

feste Größe N	`bitset<`N`>`	
aus Zahl	`bitset(`*ulong*`)`	
aus `string`	`bitset(`s`,pos,n,'0','1')`	
Bits setzen	b`.set()`	b`.set(i)`
löschen	b`.reset()`	b`.reset(i)`
negieren	b`.flip()`	b`.flip(i)`
gesetzte Bits	b`.count()`	
	b`.any()`	b`.none()`
Konversion	b`.to_string()`	
	b`.to_ulong()`	

Wrapper

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

Funktoren `<functional>`

Objektklassen mit überladenem `operator()`

einstellig	`unary_function<Arg,Res>`
$f(x) \mapsto$`-`x	`negate<T>`
$f(x) \mapsto$`!`x	`logical_not<T>`
zweistellig	`binary_function<`A_1`,`A_2`,Res>`
$f(x,y) \mapsto x$`+`y	`plus<T>`
$f(x,y) \mapsto x$`-`y	`minus<T>`
$f(x,y) \mapsto x$`*`y	`multiplies<T>`
$f(x,y) \mapsto x/y$	`divides<T>`
$f(x,y) \mapsto x$`%`y	`modulus<T>`
$f(x,y) \mapsto x$`==`y	`equal_to<T>`
$f(x,y) \mapsto x$`!=`y	`not_equal_to<T>`
$f(x,y) \mapsto x > y$	`greater<T>`
$f(x,y) \mapsto x < y$	`less<T>`
$f(x,y) \mapsto x >= y$	`greater_equal<T>`
$f(x,y) \mapsto x <= y$	`less_equal<T>`
$f(x,y) \mapsto x$`&&`y	`logical_and<T>`
$f(x,y) \mapsto x$`||`y	`logical_or<T>`
Negierer/Binder	
$f(args) \mapsto$`!`$f(args)$	`not_fn(`*args*`)`
$f(args) \mapsto f(fewer)$	`bind(`f`,`*args*`)`
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	*mit Operatoren*	`* ++`
Input		`== != * -> ++`
Forward	*zusätzlich*	`=`
Bidirectional	*zusätzlich*	`-`
Random access	*zusätzlich*	`< <= > >= + -` `+= -= []`
Reverse (Bidirectional)	*arbeitet auf*	*ri*`.base()`
mit vertauschter Richtung		`++ --`
Iterator *n* Stellen weiterrücken		`advance(`*it*`,`*n*`)`
Abstand (Input-)Iteratoren		`distance(`*f*`,`*l*`)`

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

Iterator-Adapter

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

Beispiel:

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

Ausgabestrom-Iteratoren	`ostream_iterator<`*T*`>`
Konstruktor	*o*`(`*strom*`,`*trenner*$_{opt}$`)`

Beispiel:

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

Eingabestrom-Iteratoren	`istream_iterator<`*T*`>`
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 *p*	`string_view(`*p*`)`
n Zeichen	`string_view(`*p*`, `*n*`)`
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 *s*`[`*i*`]`	*s*`.copy(`*p*`,`*n*`,`*i*`=0)`
nach char-Feld *p*	*max.* `n` *Zeichen, ohne* `'\0'`
`&s[0]`	*s*`.data()` *nicht terminiert!*
n Zeichen	*s*`.remove_prefix(n)`
entfernen	*s*`.remove_suffix(n)`

Zeichenketten <string>

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

Zeichenarten <cctype>

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

Reguläre Ausdrucke <regex>

Raw string s	`R"`*delim*`(...)`*delim*`"`
Konstruktor	`regex(`s`,`$type_{opt}$`)`
Typ	`ECMAScript,basic,grep,`...
Übereinstimmung	`regex_match(`s`,`rex`)`
Suchergebnis	`regex_match` m
Suche	`regex_search(`s`,`m`,`rex`)`
gesamt	m`[0]`
Teile	m`[1]`...m`[`m`.size()-1]`
Ersetzen	`regex_replace(`s`,`rex`,`new`)`

Mathematik <cmath>

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

Komplexe Zahlen <complex>

Spezialisierungen	`complex<float>`
	`complex<double>`
	`complex<long double>`
Realteil	c`.real()` `real(`c`)`
Imaginärteil	c`.imag()` `imag(`c`)`
Betrag r	`abs(`c`)`
Winkel ϕ	`arg(`c`)`
Betragsquadrat	`norm(`c`)`
Konjugierte	`conj(`c`)`
	`polar(`r`,`ϕ`)`
Funktionen aus	`<cmath>`

Zahlen-Wertebereiche <limits>

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

Hilfsfunktionen <cstdlib>

Kommando	`system(`*befehl*`)`
Programmende	`exit(`*fehlerNr*`)`

Dateisystem <filesystem>

in `namespace std::filesystem` (Auswahl)

Pfade	`path(`s`)`
vergleichen	`< ... == !=`
verketten	*p/=p2* *p/p2*
zerlegen	p`.root_name()`
	p`.root_directory()`
	p`.root_path()`
	p`.relative_path()`
	p`.parent_path()` p`.filename()`
	p`.stem()` p`.extension()`
abfragen	`current_path()` `exists(`p`)`
	`is_directory(`p`)` `file_size(`p`)`
anlegen, kopieren	`create_directories(`p`)`
	`copy(`*from,to*`)`
Pfad durchlaufen	`directory_iterator(`p`)`
(auch rekursiv)	`recursive_`...
löschen	`remove(`p`)` `remove_all(`p`)`
Ausnahme	`filesystem_error`

Ein-/Ausgabeströme <iostream>

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

Formatierung mit Manipulatoren <iomanip>

Eingabe	*is*>>*manip*
Ganzzahlbasis	dec hex oct
Leerraum (nicht) übergehen	ws *no*skipws
Ausgabe	***os*<<*manip***
Zeilenvorschub	endl
Puffer leeren	flush
logische Werte	*no*boolalpha
Zahldarstellung	*no*showpos *no*showpoint
Nichtganzzahlen	fixed scientific
Genauigkeit	setprecision(*n*)
Ganzzahlbasis	dec hex oct *no*showbase
Hexziffern, e/E	*no*uppercase
Ausgabebreite	setw(*n*) *(flüchtig)*
Ausrichtung	left internal right
Füllzeichen	setfill(*c*)
Zeit ausgeben	put_time(*tptr*, "*format*")

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

Stringströme <sstream>

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

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

Zeitfunktionen <ctime>

Systemzeit	time(*tptr*) *(Sek. ab 1970)*
Zeitdifferenz	difftime(t_2,t_1)
lokale und	localtime(*tptr*)
Weltzeit	gmtime(*tptr*)
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(*tmptr*)
Zeichenkette	asctime(*tmptr*)
	ctime(*tptr*)

Uhren und Zeitspannen <chrono>

Uhren	high_resolution_clock
	steady_clock system_clock
Zeitpunkt	*uhr*::now()
konvertieren	system_clock::to_time_t(*t*)
	system_clock::from_time_t(*t*)
Zeitspannen	*t1*.time_since_epoch()
berechnen	*t2*>=*t1* *t2*-*t1* *t1*+*dt*
Ticks	*dt*.count()
Zeiteinheiten	duration<*rep*, *ratio*>
	nanoseconds ... minutes hours
umrechnen	duration_cast<*Ziel*>(*dt*)
gebrochen	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
je nach intmax_t	peta exa *zetta yotta*
chrono_literals	2h+3min+4ms+5us