

- **Explizite arithmetische Typumwandlungen bzw. implizite Typumwandlungen bei Zuweisung oder Argumentübergabe während eines Funktionsaufrufs**

Neben den bereits erwähnten impliziten Umwandlungen werden auch die nachfolgenden Typumwandlungen durchgeführt. Entsprechendes gilt beim Einsetzen von Argumenten in Funktionen, wenn der Argumenttyp vom Parametertyp abweicht.

ganzzahlig	→	ganzzahlig mit Vorzeichen	Wert unverändert, falls im Zielbereich darstellbar; sonst implementierungsabhängig
ganzzahlig	→	ganzzahlig ohne Vorzeichen	kleinster Wert modulo $2^n$
ganzzahlig, reell	→	reell	nächstkleinerer oder -größerer Gleitpunktwert, falls Ausgangswert innerhalb des Zielbereichs; sonst undefiniert
reell	→	ganzzahlig	Abschneiden des Dezimalbruchs, falls Ergebnis im Zielbereich darstellbar; sonst undefiniert

Explizite Typumwandlungen können mit dem bereits in C vorhandenen Cast-Operator

→  $(\text{Typ})$  → unärer Ausdruck →

oder sicherer durch

→ `static_cast` →  $\langle$  → Typ →  $\rangle$  →  $($  → Ausdruck →  $)$  →

erfolgen.

- **Mathematische Standardfunktionen (Auszug)**

#### Funktionen in <cmath>

Im folgenden stehen  $x$  und  $y$  für Argumente desselben Gleitpunktzahltyps, der auch Ergebnistyp ist,  $n$  für ein Argument vom Typ `int` und  $l$  für ein Argument vom Typ `long`.

Funktionen	Bedeutung
<code>abs(x)</code> <code>fabs(x)</code>	$ x $
<code>sqrt(x)</code>	$\sqrt{x}$
<code>sin(x)</code> <code>cos(x)</code> <code>tan(x)</code>	trigonometrische Funktionen
<code>asin(x)</code> <code>acos(x)</code> <code>atan(x)</code>	trigonometrische Umkehrfunktionen
<code>exp(x)</code>	$e^x$
<code>sinh(x)</code> <code>cosh(x)</code> <code>tanh(x)</code>	hyperbolische Funktionen
<code>log(x)</code> <code>log10(x)</code>	$\ln(x)$ $\lg(x)$
<code>floor(x)</code> <code>ceil(x)</code>	$[x]$ $-[-x]$
<code>pow(x,y)</code> <code>pow(x,n)</code>	$x^y$ $x^n$
<code>atan2(y,x)</code>	Argumentfunktion mit Wertebereich $[-\pi, \pi]$
<code>ldexp(x,n)</code>	$x \cdot 2^n$

#### Funktionen in <cstdlib>

Funktionen	Bedeutung	Bemerkung
<code>abs(n)</code>	$ n $	Ergebnistyp <code>int</code>
<code>abs(l)</code> <code>labs(l)</code>	$ l $	Ergebnistyp <code>long</code>
<code>rand()</code> <code>srand(n)</code>	Pseudozufallszahl zwischen 0 und <code>RAND_MAX</code>	Ergebnistyp <code>int</code>

In C++11 sind weitere Pseudozufallszahlengeneratoren mit unterschiedlichen Verteilungsfunktionen verfügbar (<random>).

- Beispiele zur Fehlerbehandlung mit `errno`

*Programm:*

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cerrno>
#include <cstring>

using namespace std;

int main()
{
    errno = 0;
    cout << "sqrt(-1) = " << sqrt(-1) << endl;
    cout << strerror(errno) << endl << endl;

    errno = 0;
    cout << "exp(10000) = " << exp(10000) << endl;
    cout << strerror(errno) << endl << endl;

    errno = 0;
    cout << "log(0) = " << log(0) << endl;
    cout << strerror(errno) << endl << endl;

    errno = 0;
    cout << "atan2(0,0) = " << atan2(0,0) << endl;
    cout << strerror(errno) << endl << endl;

    return(0);
}
```

*BildschirmAusgabe (g++-7.5, Ubuntu Linux 18.04):*

```
sqrt(-1) = -nan
Numerical argument out of domain
```

```
exp(10000) = inf
Numerical result out of range
```

```
log(0) = -inf
Numerical result out of range
```

```
atan2(0,0) = 0
Success
```

## • Logische Ausdrücke

Wertebereich des Datentyp `bool`: `true`, `false`.

Die logischen Operatoren `&&` (und), `||` (oder) und `!` (nicht) akzeptieren Operanden vom Datentyp `bool` Datentypen und liefern ein Ergebnis vom Datentyp `bool`.

Die Auswertung der binären Operatoren `&&` und `||` erfolgt von links nach rechts. Der zweite Operand wird nur ausgewertet, wenn das Ergebnis nach der Auswertung des ersten Operanden noch nicht feststeht.

Bsp.: `if (x>0 && log(x)>y) { ... }` Auch die Vergleichsoperatoren `<`, `>`, `<=`, `>=`, `==` und `!=` liefern als Ergebnis den Datentyp `bool`.

Wird ein arithmetischer Ausdruck an einer Stelle eingesetzt, an der ein Wert vom Datentyp `bool` erwartet wird, so erfolgt eine implizite Typumwandlung in `bool`. Dabei wird ein arithmetischer Wert ungleich 0 in `true` und 0 in `false` umgewandelt. Das trifft beispielsweise zu, wenn ein arithmetischer Ausdruck als Bedingung in einer `if`-Anweisung oder einer Wiederholungsanweisung verwendet wird. Dasselbe gilt für die logischen Operatoren `&&`, `||` und `!`.

(Umgekehrt wird bei Typumwandlungen boolescher Werte in Zahlen `false` in 0 und `true` in 1 konvertiert.)

```
Vorsicht:  if (i==1) { ... }           // falls i den Wert 1 hat
            if (i=1)  { ... }           // immer
            if (1<=x && x<=2) { ... }    // falls x in [1,2]
            if (1<=x<=2) { ... }        // immer
            if (!(x>0)) { ... }         // falls x<=0
            if (!x>0) { ... }           // falls x==0
```

Mit dem ternären Operators `? :` läßt sich der bedingte Ausdruck

*Bedingung?* *Ausdruck1* : *Ausdruck2*

bilden. Falls die Bedingung einen Wert  $\neq 0$  liefert, wird nur der Ausdruck1 ausgewertet und ist das Ergebnis des bedingten Ausdrucks. Andernfalls wird der Ausdruck2 entsprechend herangezogen.

```
Bsp.:  c = a>b ? a : b    /* c=max(a,b) */
        c = a>=0 ? a : -a /* c=abs(a)   */
```

## • Bitoperatoren

Bitoperatoren können auf ganzzahlige Operanden angewandt werden. In der Regel sollten diese einen vorzeichenlosen Datentyp (z.B. `unsigned`) haben, weil die Wirkung einzelner Operatoren auf vorzeichenbehaftete Datentypen (z.B. `int`) implementationsabhängig ist.

Operator	Bedeutung
<code>~</code>	Bitweise nicht
<code>&lt;&lt; &gt;&gt;</code>	Linksshift, Rechtsshift
<code>&amp;</code>	Bitweise und
<code>^</code>	Bitweise exklusives oder (xor)
<code> </code>	Bitweise (inklusive) oder
<code>&lt;&lt;= &gt;&gt;= &amp;=  = ^=</code>	Bitzuweisungen

Bsp.:

```
unsigned v=14, b;
b = (v>>2)&1u; /* Bit 2 von v (von rechts ab 0 gezaehlt) in b schreiben */
               /* v unveraendert                                         */
v |= 1u<<4; /* Bit 4 in v (von rechts ab 0 gezaehlt) auf 1 setzen */
v &= ~(1u<<3); /* Bit 3 in v (von rechts ab 0 gezaehlt) auf 0 setzen */
```

## Vorrang der Operatoren

Die folgende Tabelle enthält die Operatoren von C++ nach ihrer Bindungsstärke geordnet.  
(Weiter oben aufgeführte Operatoren binden stärker.)

Die Tabelle gibt die Intention der Syntax wieder, in Einzelfällen kann es zu Abweichungen kommen.

<i>Operatoren</i>	<i>Assoziativität</i>	<i>Bemerkungen</i>
::	links	
() [] -> . ++ --	links	++ -- postfix
typeid static_cast const_cast	rechts	Vorrang wie vorige Zeile
dynamic_cast reinterpret_cast	rechts	Vorrang wie vorige Zeile
++ -- + - ! ~ & * sizeof new delete ( <i>Typ</i> )	rechts	++ -- präfix + - & * unär
.* ->*	links	
* / %	links	* binär
+ -	links	+ - binär
<< >>	links	
< > <= >=	links	
== !=	links	
&	links	& binär
^	links	
	links	
&&	links	
	links	
? :	rechts	ternär
= += -= *= /= %= &= ^=  = <<= >>=	rechts	
throw	rechts	
,	links	