Wertzuweisungen

Durch eine Wertzuweisung wird einer Variablen ein Wert zugeordnet.
Das =-Zeichen wird Zuweisungsoperator genannt.

Syntax:

```
<Variablenname> =  <Ausdruck> ;
```

`<Ausdruck>` steht zum Beispiel für ein Literal

```
c = 2;
```

oder für einen Term, also einen berechneten Ausdruck

```
mittelwert = (zahl1 + zahl2) / 2;
umfang = 2 * 3.14 * radius;
```

Der Typ der Variablen sollte mit dem Ergebnistyp des Ausdrucks übereinstimmen. C++ überprüft die Übereinstimmung der Typen nicht konsequent. Dadurch können falsche Ergebnisse entstehen.

Für einfache Zuweisungen mit einem Rechen-Operator (+,-,*,/)

```
<variable> = <variable> <Operator> <Ausduck>
```

gibt es eine Kurzform der Art

```
<variable> <operator>= <Ausduck>
```

Beispiel:

```
i = i + 1;  /*Normale Form */
i += 1;     /* Kurzform    */
```

Ausdrücke

Arithmetische Ausdrücke

Für arithmetische Operationen gibt es die folgenden Operatoren:

arithmetische Operation	**mathematische Schreibweise**	**C++-Schreibweise**
Addition	a + b	a **+** b
Subtraktion	a - b	a **-** b
Multiplikation	a • b	a ***** b
Division mit Gleitkommazalen	a : b	a **/** b
Ganzzahldivision ohne Rest	a	a **/** b
Rest einer Ganzzahldivision (Modulo)		a **%** b

Alle anderen mathematischen Funktionen werden durch Funktionsaufrufe berechnet. Die Deklarationen der Funktionen finden Sie in der Headerdatei **cmath**. Man muss also am Programmanfang `#include <cmath>` schreiben, um sie benutzen zu können.

Funktion	**mathematische Schreibweise**	**C++-Schreibweise**
Potenzieren	a^b	`pow ( a , b)`
Wurzelziehen	$\sqrt{a}$	`sqrt( a )`
Sinus	sin a	`sin( a)`
Kosinus	cos a	`cos( a )`
nat. Logarithmus	ln a	`log( a )`
e-Funktion	e^a	`exp( a )`
Arcus-Tangens	arctan (x)	`atan (x)`
Betrag (int-Wert)	|a|	`abs (a)`
Betrag (float-Wert)	|a|	`fabs (a)`
Abrunden auf ganze Zahl	⌊a⌋	`floor(a)`

Zusätzlich kennt C++ die **Inkrement- und Dekrementoperatoren**.

`++x`	x wird um den Wert 1 erhöht, bevor x im Ausdruck weiterverwendet wird.
`--x`	x wird um den Wert 1 erniedrigt, bevor x im Ausdruck weiterverwendet wird.
`x++`	x wird inkrementiert, nachdem x im Ausdruck verwendet wurde.
`x--`	x wird dekrementiert, nachdem es im Ausdruck verwendet wurde.

Beispiele:

```
/* Variablendefinitionen */
int i=1, j=1;

/* Anweisungen */
cout << i++ << endl; // Ausgabe 1, neuer Wert von i 2
cout << ++j << endl; // Ausgabe 2, neuer Wert von j 2
```

Vergleiche und logische Ausdrücke

Es können Ausdrücke (keine Strings, Arrays etc.) des gleichen Typs miteinander verglichen werden. Folgende Vergleichsoperatoren stehen zur Verfügung:

Vergleich	**C++-Schreibweise**
a gleich b	`a == b`
a größer b	`a > b`
a kleiner b	`a < b`
a größer oder gleich b	`a >= b`
a kleiner oder gleich b	`a <= b`
a ungleich b	`a != b`

Beispiele (Codeausschnitte):

```
(b * b) < 4
d != 0
a  <  b
```

Das Ergebnis des Vergleichs ist vom Typ `bool` und kann den Wert `true` oder `false` annehmen. Intern werden für die Darstellung dieser Werte Zahlen verwendet: 0 für falsch und 1 oder andere Zahlen ungleich 0 für wahr.

Logische Ausdrücke können durch die Operatoren `||` (logisches ODER) und `&&` (logisches UND) miteinander verknüpft werden. Ein logisches NICHT erhält man mit einem Ausrufezeichen (`!`).

Man kann in logischen Ausdrücken mit runden Klammern wie in der Mathematik die Reihenfolge der Auswertung bestimmen.

Bitausdrücke

Für die Manipulation einzelner Bits gibt es in C++ die folgenden Operatoren:

Ausdruck	Wirkung
`x << y`	x wird um y-Bits nach links geschoben
`x >> y`	x wird um y-Bits nach rechts geschoben
`x & y`	Bitweises UND von x und y
`x | y`	Bitweises ODER von x und y
`x ^ y`	Bitweises XOR von x und y
`~ x`	Einerkomplement von x

Bitoperatoren werden hauptsächlich in der hardwarenahen Programmierung verwendet.

Die Operatoren der Bitmanipulation haben eine geringere Priorität als die Vergleichsoperatoren. Bei Vergleichen müssen daher Ausrücke mit Bitoperationen in Klammern gesetzt werden.

Beispiel:

```
if ((i& 0Xf0) == true)        /* Sind die oberen 4 Bit gesetzt ?? */
```