C-Programmierung: Ausdrücke und Operatoren#Division .2F

C-Programmierung: Ausdrücke und Operatoren#Division .2F

09-11-2013

Table of Contents

C-Programmierung: Ausdrücke und Operatoren#Division .2F

Ausdrücke[Bearbeiten]

Ein Ausdruck ist eine Kombination aus Variablen, Konstanten, Operatoren und Rückgabewerten von Funktionen. Die Auswertung eines Ausdrucks ergibt einen Wert.

Operatoren[Bearbeiten]

Man unterscheidet zwischen unären und binären Operatoren. Unäre Operatoren besitzen einen Operanden, binäre Operatoren besitzen deren zwei. Die Operatoren *, &, + und – kommen sowohl als unäre wie auch als binäre Operatoren vor.

Vorzeichenoperatoren[Bearbeiten]

Negatives Vorzeichen -[Bearbeiten]

Liefert den negativen Wert eines Operanden. Der Operand muss ein arithmetischer Typ sein. Beispiel:

```
printf("-3 minus -2 = %i", -3 - -2); // Ergebnis ist -1
```

Positives Vorzeichen +[Bearbeiten]

Der unäre Vorzeichenoperator `+` wurde in die Sprachdefinition aufgenommen, damit ein symmetrischer Operator zu – existiert. Er hat keine Einfluss auf den Operanden. So ist beispielsweise `+4.35` äquivalent zu `4.35` . Der Operand muss ein arithmetischer Typ sein. Beispiel:

```
printf("+3 plus +2= %i", +3 + +2); // Ergebnis ist 5
```

Arithmetik[Bearbeiten]

Alle arithmetischen Operatoren, außer dem Modulo-Operator, können sowohl auf Ganzzahlen als auch auf Gleitkommazahlen angewandt werden. Arithmetische Operatoren sind immer binär.

Beim + und − Operator kann ein Operand auch ein Zeiger sein, der auf ein Objekt (etwa ein Array) verweist und der zweite Operand ein Integer sein. Das Resultat ist dann vom Typ des Zeigeroperanden. Wenn P auf das *i*-te Element eines Arrays zeigt, dann zeigt P + n auf das *i*+*n*-te Element des Array und P - n zeigt auf das *i*-*n*-te Element. Beispielsweise zeigt P + 1 auf das nächste Element des Arrays. Ist P bereits das letzte Element des Arrays, so verweist der Zeiger auf das nächste Element nach dem Array. Ist das Ergebnis nicht mehr ein Element des Arrays oder das erste Element nach dem Array, ist das Resultat undefiniert.

Addition +[Bearbeiten]

Der Additionsoperator liefert die Summe der Operanden zurück. Beispiel:

```
int a = 3, b = 5;
int ergebnis;
ergebnis = a + b; // ergebnis hat den Wert 8
```

Subtraktion -[Bearbeiten]

Der Subtraktionsoperator liefert die Differenz der Operanden zurück. Beispiel:

```
int a = 7, b = 2;
int ergebnis;
ergebnis = a - b; // ergebnis hat den Wert 5
```

Wenn zwei Zeiger subtrahiert werden, müssen beide Operanden Elemente desselben Arrays sein. Das Ergebnis ist vom Typ `ptrdiff` . Der Typ `ptrdiff` ist ein vorzeichenbehafteter Integerwert, der in der Headerdatei `<stddef.h>` definiert ist.

Multiplikation *[Bearbeiten]

Der Multiplikationsoperator liefert das Produkt der beiden Operanden zurück. Beispiel:

```
int a = 5, b = 3;
int ergebnis;
ergebnis = a * b; // variable 'ergebnis' speichert den Wert 15
```

Division /[Bearbeiten]

Der Divisionsoperator liefert den Quotienten aus der Division des ersten durch den zweiten Operanden zurück. Beispiel:

```
int a = 8, b = 2;
int ergebnis;
ergebnis = a/b; // Ergebnis hat den Wert 4
```

Bei einer Division durch 0 ist das Resultat undefiniert.

Aufgepasst bei Gleitkommazahlen
Funktionen wie 'printf' erwarten bei Berechnungen als Argumente ganze Zahlen. Darum führt eine einfache Division wie 3/5 als Argument zu einem falschen Resultat. Der Funktion müssen zwingend Gleitkommazahlen übergeben werden:

```
printf("3/5 = %f", 3.0/5.0 ); //Ergebnis ist 0.600000
```

Modulo %[Bearbeiten]

Der Modulo-Operator liefert den Divisionsrest. Die Operanden des Modulo-Operators müssen vom ganzzahligen Typ sein. Beispiel:

```
int a = 5, b = 2;
int ergebnis;
ergebnis = a % b; // Ergebnis hat den Wert 1
```

Ist der zweite Operand eine 0, so ist das Resultat undefiniert.

Zuweisung[Bearbeiten]

Der linke Operand einer Zuweisung muss ein modifizierbarer L-Wert sein.

Zuweisung =[Bearbeiten]

Bei der einfachen Zuweisung erhält der linke Operand den Wert des rechten. Beispiel:

```
int a = 2, b = 3;
a = b; //a erhaelt Wert 3
```

Kombinierte Zuweisungen[Bearbeiten]

Kombinierte Zuweisungen setzen sich aus einer Zuweisung und einer anderen Operation zusammen. Der Operand

```
 a += b
```

wird zu

```
 a = a + b
```

erweitert. Es existieren folgende kombinierte Zuweisungen:

`+=` , `-=` , `*=` , `/=` , `%=` , `&=` , `|=` , `^=` , `<<=` , `>>=`

Inkrement ++[Bearbeiten]

Der Inkrement-Operator erhöht den Wert einer Variablen um 1. Wird er auf einen Zeiger angewendet, erhöht er dessen Wert um die Größe des Objekts, auf das der Zeiger verweist. Man unterscheidet Postfix (a++)- und Präfix (++a)-Notation. Bei der Postfix-Notation wird die Variable inkrementiert, nachdem sie verwendet wurde. Bei der Präfix-Notation wird sie inkrementiert, bevor sie verwendet wird. Die Notationsarten unterscheiden sich durch ihre Priorität (siehe Liste der Operatoren, geordnet nach ihrer Priorität). Der Operand muss ein L-Wert sein.

Dekrement --[Bearbeiten]

Der Dekrement-Operator verringert den Wert einer Variablen um 1. Wird er einen auf Zeiger angewendet, verringert er dessen Wert um die Größe des Objekts, auf das der Zeiger verweist. Auch hier unterscheidet man Postfix- und Präfix-Notation.

Vergleiche[Bearbeiten]

Das Ergebnis eines Vergleichs ist 1, wenn der Vergleich zutrifft, andernfalls 0. Als Rückgabewert liefert der Vergleich einen integer-Wert. In C wird der boolsche Wert *true* durch einen Wert ungleich 0 und *false* durch 0 repräsentiert. Beispiel:

```
a = (4 == 3); // a erhaelt den Wert 0
a = (3 == 3); // a erhaelt den Wert 1
```

Gleichheit ==[Bearbeiten]

Der Gleichheits-Operator vergleicht die beiden Operanden auf Gleichheit. Er besitzt einen geringeren Vorrang als <, >, <= und >=.

Ungleichheit !=[Bearbeiten]

Der Ungleichheits-Operator vergleicht die beiden Operanden auf Ungleichheit. Er besitzt einen geringeren Vorrang als <, >, <= und >=.

Kleiner <[Bearbeiten]

Der Kleiner als-Operator liefert dann 1, wenn der Wert des linken Operanden kleiner ist als der des Rechten. Beispiel:

```
int a = 7, b = 2;
int ergebnis;
ergebnis = a < b; // Ergebnis hat den Wert 0
ergebnis = b < a; // Ergebnis hat den Wert 1
```

Größer >[Bearbeiten]

Der Größer als-Operator liefert dann 1, wenn der Wert des linken Operanden größer ist als der des Rechten. Beispiel:

```
int a = 7, b = 2;
int ergebnis;
ergebnis = a > b; // Ergebnis hat den Wert 1
ergebnis = b > a; // Ergebnis hat den Wert 0
```

Kleiner gleich <=[Bearbeiten]

Der Kleiner gleich-Operator liefert dann 1, wenn der Wert des linken Operanden kleiner oder exakt gleich ist wie der Wert des Rechten. Beispiel:

```
int a = 2, b = 7, c = 7;
int ergebnis;
```

```
ergebnis = a <= b; // Ergebnis hat den Wert 1
ergebnis = b <= c; // Ergebnis hat ebenfalls den Wert 1
```

Größer gleich >=[Bearbeiten]

Der Größer Gleich - Operator liefert dann 1, wenn der Wert des linken Operanden größer oder exakt gleich ist wie der Wert des Rechten. Beispiel:

```
int a = 2, b = 7, c = 7;
int ergebnis;
ergebnis = b >= a; // Ergebnis hat den Wert 1
ergebnis = b >= c; // Ergebnis hat ebenfalls den Wert 1
```

Aussagenlogik[Bearbeiten]

Logisches NICHT ![Bearbeiten]

Ist ein unärer Operator und invertiert den Wahrheitswert eines Operanden. Beispiel:

```
printf("Das logische NICHT liefert den Wert %i, wenn die Bedingung (nicht)
erfuellt ist.", !(2<1)); //Ergebnis hat den Wert 1
```

Logisches UND &&[Bearbeiten]

Das Ergebnis des Ausdrucks ist 1, wenn beide Operanden ungleich 0 sind, andernfalls 0. Im Unterschied zum `&` wird der Ausdruck streng von links nach rechts ausgewertet. Wenn der erste Operand bereits 0 ergibt, wird der zweite Operand nicht mehr ausgewertet und der Ausdruck liefert in jedem Fall den Wert 0. Nur wenn der erste Operand 1 ergibt, wird der zweite Operand ausgewertet. Der `&&` Operator ist ein Sequenzpunkt: Alle Nebenwirkungen des linken Operanden müssen bewertet worden sein, bevor die Nebenwirkungen des rechten Operanden ausgewertet werden.
Das Resultat des Ausdrucks ist vom Typ `int`. Beispiel:

```
printf("Das logische UND liefert den Wert %i, wenn beide Bedingungen erfuellt
 sind.", 2 > 1 && 3 < 4); //Ergebnis hat den Wert 1
```

Logisches ODER ||[Bearbeiten]

Das Ergebnis ist 1, wenn einer der Operanden ungleich 0 ist, andernfalls ist es 0. Der Ausdruck wird streng von links nach rechts ausgewertet. Wenn der erste Operand einen von 0 verschieden Wert liefert, ist das Ergebnis des Ausdruck 1, und der zweite Operand wird nicht mehr ausgewertet. Auch dieser Operator ist ein Sequenzpunkt.
Das Resultat des Ausdrucks ist vom Typ `int` . Beispiel:

```
printf("Das logische ODER liefert den Wert %i, wenn eine der beiden Bedingungen
 erfuellt ist.", 2 > 3 || 3 < 4); // Ergebnis hat den Wert 1
```

Bitmanipulation[Bearbeiten]

Bitweises UND / AND &[Bearbeiten]

Mit dem UND-Operator werden zwei Operanden bitweise verknüpft. Die Verknüpfung darf nur für Integer-Operanden verwendet werden.

Wahrheitstabelle der UND-Verknüpfung:

b	a	a & b
falsch	falsch	falsch
falsch	wahr	falsch
wahr	falsch	falsch
wahr	wahr	wahr

Beispiel:

```
 a = 45 & 35            // a == 33
```

Bitweises ODER / OR |[Bearbeiten]

Mit dem ODER-Operator werden zwei Operanden bitweise verknüpft. Die Verknüpfung darf nur für Integer-Operanden verwendet werden.

Wahrheitstabelle der ODER-Verknüpfung:

a	b	a | b
falsch	falsch	falsch
falsch	wahr	wahr
wahr	falsch	wahr
wahr	wahr	wahr

Beispiel:

```
a = 45 | 35            // a == 47
```

Bitweises exklusives ODER (XOR) ^[Bearbeiten]

Mit dem XOR-Operator werden zwei Operanden bitweise verknüpft. Die Verknüpfung darf nur für Integer-Operanden verwendet werden.

Wahrheitstabelle der XOR-Verknüpfung:

a	b	a ^ b
falsch	falsch	falsch
falsch	wahr	wahr
wahr	falsch	wahr
wahr	wahr	falsch

Beispiel:

```
a = 45 ^ 35;            // a == 14
```

Bitweises NICHT / NOT ~[Bearbeiten]

Mit der NICHT-Operation wird der Wahrheitswert eines Operanden bitweise umgekehrt

Wahrheitstabelle der NOT-Verknüpfung:

a	~a
101110	010001

a	~a
111111	000000

Beispiel:

```
a = ~45
```

Linksshift <<[Bearbeiten]

Verschiebt den Inhalt einer Variable bitweise nach links. Bei einer ganzen nicht negativen Zahl entspricht eine Verschiebung einer Multiplikation mit 2^n, wobei n die Anzahl der Verschiebungen ist, wenn das höchstwertige Bit nicht links hinausgeschoben wird. Das Ergebnis ist undefiniert, wenn der zu verschiebende Wert negativ ist.

Beispiel:

```
y = x << 1;
```

x	y
01010111	10101110

Rechtsshift >>[Bearbeiten]

Verschiebt den Inhalt einer Variable bitweise nach rechts. Bei einer ganzen, nicht negativen Zahl entspricht eine Verschiebung einer Division durch 2^n und dem Abschneiden der Nachkommastellen (falls vorhanden), wobei n die Anzahl der Verschiebungen ist. Das Ergebnis ist implementierungsabhängig, wenn der zu verschiebende Wert negativ ist.

Beispiel:

```
y = x >> 1;
```

x	y
01010111	00101011

Datenzugriff[Bearbeiten]

Dereferenzierung *[Bearbeiten]

Der Dereferenzierungs-Operator (auch Indirektions-Operator oder Inhalts-Operator genannt) dient zum Zugriff auf ein Objekt durch einen Zeiger. Beispiel:

```
int a;
int *zeiger;
zeiger = &a;
*zeiger = 3; // Setzt den Wert von a auf 3
```

Der unäre Dereferenzierungs-Operator bezieht sich immer auf den rechts stehenden Operanden.

Jeder Zeiger hat einen festgelegten Datentyp. Die Notation

```
int *zeiger
```

mit Leerzeichen zwischen dem Datentyp und dem Inhalts-Operator soll dies zum Ausdruck bringen. Eine Ausnahme bildet nur ein Zeiger vom Typ *void*. Ein so definierter Zeiger kann einen Zeiger beliebigen Typs aufnehmen. Zum schreiben muss der Datentyp per Typumwandlung festgelegt werden.

Elementzugriff ->[Bearbeiten]

Dieser Operator stellt eine Vereinfachung dar, um über einen Zeiger auf ein Element einer Struktur oder Union zuzugreifen.

```
objZeiger->element
```

entspricht

```
(*objZeiger).element
```

Elementzugriff .[Bearbeiten]

Der Punkt-Operator dient dazu, auf Elemente einer Struktur oder Union zuzugreifen

Typumwandlung[Bearbeiten]

Typumwandlung ()[Bearbeiten]

Mit dem Typumwandlungs-Operator kann der Typ des Wertes einer Variable für die Weiterverarbeitung geändert werden, nicht jedoch der Typ einer Variable. Beispiel:

```
float f = 1.5;
int i = (int)f; // i erhaelt den Wert 1

float a = 5;
int b = 2;
float ergebnis;
ergebnis = a / (float)b; //ergebnis erhaelt den Wert 2.5
```

Speicherberechnung[Bearbeiten]

Adresse &[Bearbeiten]

Mit dem Adress-Operator erhält man die Adresse einer Variablen im Speicher. Das wird vor allem verwendet, um Zeiger auf bestimmte Variablen verweisen zu lassen. Beispiel:

```
int *zeiger;
int a;
zeiger = &a; // zeiger verweist auf die Variable a
```

Der Operand muss ein L-Wert sein.

Speichergröße sizeof[Bearbeiten]

Mit dem `sizeof`-Operator kann die Größe eines Datentyps oder eines Datenobjekts in Byte ermittelt werden. `sizeof` liefert einen ganzzahligen Wert ohne Vorzeichen zurück, dessen Typ `size_t` in der Headerdatei `stddef.h` festgelegt ist.

Beispiel:

```
int a;
int groesse = sizeof(a);
```

Alternativ kann man sizeof als Parameter auch den Namen eines Datentyps übergeben. Dann würde die letzte Zeile wie folgt aussehen:

```
int groesse = sizeof(int);
```

Der Operator `sizeof` liefert die größe in Bytes zurück. Die Größe eines `int` beträgt mindestens 8 Bit, kann je nach Implementierung aber auch größer sein. Die tatsächliche Größe kann über das Macro `CHAR_BIT`, das in der Standardbibliothek `limits.h` definiert ist, ermittelt werden. Der Ausdruck `sizeof(char)` liefert immer den Wert 1.

Wird `sizeof` auf ein Array angewendet, ist das Resultat die Größe des Arrays, `sizeof` auf ein Element eines Arrays angewendet liefert die Größe des Elements. Beispiel:

```
char a[10];
sizeof(a);    // liefert 10
sizeof(a[3]); // liefert 1
```

Der `sizeof` -Operator darf nicht auf Funktionen oder Bitfelder angewendet werden.

Sonstige[Bearbeiten]

Funktionsaufruf ()[Bearbeiten]

Bei einem Funktionsaufruf stehen nach dem Namen der Funktion zwei runde Klammern. Wenn Parameter übergeben werden, stehen diese zwischen diesen Klammern. Beispiel:

```
funktion(); // Ruft funktion ohne Parameter auf
funktion2(4, a); // Ruft funktion2 mit 4 als ersten und a als zweiten Parameter
 auf
```

Komma-Operator ,[Bearbeiten]

Der Komma-Operator erlaubt es, zwei Ausdrücke auszuführen, wo nur einer erlaubt wäre. Die Ergebnisse aller durch diesen Operator verknüpften Ausdrücke außer dem letzten werden verworfen. Am häufigsten wird er in For-Schleifen verwendet, wenn zwei Schleifen-Variablen vorhanden sind.

```
int x = (1,2,3); // entspricht  int x = 3;
for (i=0,j=1; i<10; i++, j--)
{
   //...
}
```

Bedingung ?:[Bearbeiten]

Der Bedingungs-Operator hat die Syntax

```
Bedingung ? Ausdruck1 : Ausdruck2
```

Zuerst wird die Bedingung ausgewertet. Trifft diese zu, wird der erste Ausdruck abgearbeitet, andernfalls der zweite. Beispiel:

```
 int a, b, max;
a = 5;
b = 3;
max = (a > b) ? a : b; //max erhalt den Wert von a (also 5),
                       //weil diese die Variable mit dem größeren Wert ist
```

Indizierung [][Bearbeiten]

Der Index-Operator wird verwendet, um ein Element eines Arrays anzusprechen. Beispiel:

```
 a[3] = 5;
```

Klammerung ()[Bearbeiten]

Geklammerte Ausdrücke werden vor den anderen ausgewertet. Dabei folgt C den Regeln der Mathematik, dass innere Klammern zuerst ausgewertet werden. So durchbricht

```
 ergebnis = (a + b) * c
```

die Punkt-vor-Strich-Regel, die sonst bei

```
 ergebnis = a + b * c
```

gelten würde.