Informatik 1

Ausdrücke und Operatoren

Ausdrücke

- Ausdrücke bestehen aus Variablen, Literalen, Operatoren und Funktionsaufrufen.
- Klassifikation der Operatorentypen nach
  - Stellung des Operator und
  - Anzahl Operanden

Stellung	Anzahl Operanden	Beispiel
Infix	2 (binär)	a + b
	3 (ternär)	(a > b) ? a: b
Postfix	1 (unär)	a++
Präfix	1 (unär)	-a

- Operatoren verknüpfen die Werte der Operanden.
- Die Operanden sind ebenfalls als Ausdruck gegeben

Operatoren (1 / 3)

Vorrang	Operator	Beschreibung	Beispiel
1 Postfix	++ -- [ ]	Postinkrement Postdekrement Indexoperator	a++ i— a[i] a[7]
2 Präfix	++ -- + - ~ ! ( )	Inkrement Dekrement Plus Negation Bitweise Komplement Boolesche Negation Cast	++a --i +a +7 -x -8.5 ~a ~15 ! b ! true (int) a (int) 7.15
3	* / %	Multiplikation Division ohne Rest Modulo, Division mit Rest	a * b 5 * 7 a / b 5.0 / 2.0 a % b 7 % 2
4	+ -	Addition Subtraktion	a + b 5 + 8 a – b 6.0 – 12.5

Operatoren (2 / 3)

Vorrang	Operator	Beschreibung	Beispiel
5	<< >>> >>	Bits links nach links verschieben Rechts-Shift Vorzeichenbit wird nachgeschoben	a << i a << 5 a >> i 15 >> 2 a >>> 2 15 >>> 2
6	< > <= >= instanceof	Kleiner Größer Kleiner oder gleich Größer oder gleich Datentyp prüfen	a < b 5 < 11 a > b 5.0 > 11.0 a <= b 11 <= 5 a >= b 7 > 11 a instanceof String
7	== !=	Identität Nicht identisch	a == b 5 == 7 a != b 5.0 != 2.0

Operatoren (3 / 3)

Vorrang	Operator	Beschreibung	Beispiel
8	&	Logisches UND Bitweise UND	a & b true & false 5 & 7
9	^	Logisches XOR Bitweise XOR	a ^ b true ^ false -1 ^ 8
10	|	Logisches ODER Bitweise ODER	a | b true | false 7 | 128
11	&&	Kurzschlussoperator, logisches UND	a && b a && true
13	||	Kurzschlussoperator, ODER	a || b false || b
14	? :	Vergleichsoperator	b ? c : d (a > 7 : true : false)

Ausdrücke Syntax

- Ein Literal, eine Variable oder ein Funktionsaufruf ist ein Ausdruck.
- Wenn A und B Ausdrücke sind, dann sind folgendes wieder Ausdrücke:

  (A) A Op1 B Op2 A A Op3

  wobei Op1 ein binärer Operator Op2 ein unärer Präfix und Op3 ein unärer Postfix Operator ist.

Beispiel für Ausdrücke mit int a = 7;

a (a) a + (2 * a)

a++ -1 + 7 * 12 / (a - 1)

- Keine Ausdrücke:

  a + * 2 (* kein Vorzeichen, + kein nachgestellter unärer Operator

  a + (2 + a)) schließende Klammer zu viel

Ausdrücke Semantik

1. Ein Ausdruck und dessen Teilausdrucke besitzen immer jeweils einen definierten Datentyp.
2. Bei arithmetischen Ausdrücken werden, niederwertige Datentypen ggf. zu höherwertigen Datentypen konvertiert (widening conversion, Details später)
3. Der Datentyp bestimmt die Ausführungsart des Operators, z.B. ganzzahlige Division oder Gleitkommadivision
4. Es gibt Vorrangsregeln, die durch die Operatoren definiert sind. Es gelten Assoziativitätsregeln.
5. Ausdrücke werden von links nach rechts zur Laufzeit ausgewertet.

Ausdrücke Semantik (1)

- Literale bzw. Variablen haben "ihren" Datentyp

  1 (int) 7.0 (double)

  ‘a‘ (char) true (boolean)

- Der Datentyp eines Ausdrucks ist der Ergebnisdatentyp der zuletzt durchgeführten Operation

  5 > 7 (boolean)

  Vergleichsoperatoren sind immer boolean im Ergebnis

- Der Datentyp eines arithmetischen Ausdrucks ist immer der höherwertigere Datentyp des linke oder rechten Teilausdrucks
- Höherwertig: byte < short < int < long < float < double

  1 + 7.0 (double) 1L * 7.0f (float)

Ausdrücke (2)

- Werte vom niederwertigeren Datentyp werden zum höherwertigen automatisch konvertiert.
- Entweder vom Compiler oder erst zur Laufzeit (Regeln später).
- f + 1 Compiler konvertiert int Wert 1 in den Gleitkommawert 1.0f
- f + i Zur Laufzeit wird der Wert i zur entsprechenden Gleitkommazahl konvertiert

Ausdrücke (3)

- Die Art der Operation: ganzzahlige bzw. Gleitkomma hängt vom Datentyp der Operanden ab (höherwertige)

  1 + 1L (long Addition)

  1.0 + 5 (double Addition)
- % ist Rest der Division

  7 % 2 ist 1   7 / 2 i st 3

  7.0 / 2.0   ist 3.5
- (% ist auch für Gleitkommazahlen und negative Zahlen definiert, aber nicht relevant)
- % ist überladen, d.h. er existiert für verschiedene Datentype (6 in Java)

Ausdrücke (4)

- Analog der Punkt- vor Strichrechnung haben Operatoren einen genau definierten Vorrang untereinander
- Vorrang ist aus der Operatorentabelle ersichtlich (oben höchster Vorrang)
- Vorzeichen haben einen höheren Vorrang als binäre Operatoren
  - a * 7 entspricht (-a) * 7 und nicht - (a * 7)
- Unäre Postfix-Operatoren vor Präfix

  -a++ entspricht -(a++)
- Binäre vor ternären Operator

  a > b ? a : b entspricht (a > b) ? a : b

- **Ausdrücke immer vollständig Klammern!**

Ausdrücke (4)

- Assoziativität
  - Binäre und ternäre Operatoren (außer Zuweisungsoperatoren) sind links-assoziativ.
  - Unäre Operatoren und Zuweisungsoperatoren sind rechts-assoziativ (außer Postinkrement, -dekrement).

a + b + c + d entspricht ((a + b) + c) + d

- - - - a entspricht - (-(-(-a)))

a = b = c = d entspricht a = (b = (c = d))

4 / 2 * 3 entspricht (4 / 2) * 3

Ausdrücke (5)

- Ausführungsreihenfolge
  - **Von links nach rechts** unter Berücksichtigung des Vorrangs der Operatoren, der Assoziativitätsregeln und der Klammerung

    A + B -> A wird zuerst ausgewertet
- In der Programmiersprache C

  Nicht definiert

  B könnte zuerst ausgewertet werden

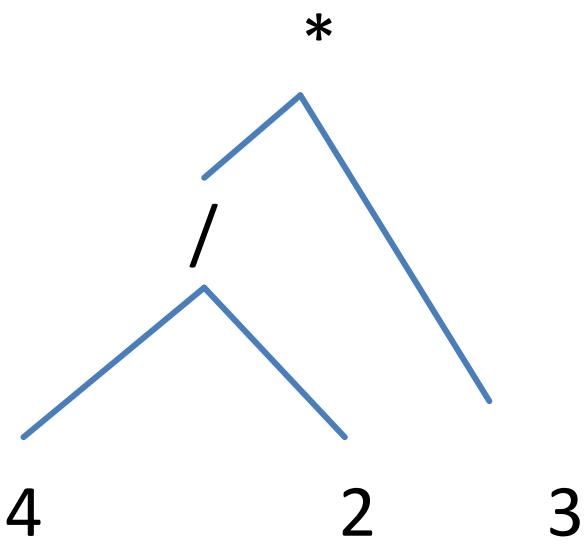
Syntaxbaum

- Syntaxbaum: Interne Repräsentation eines Ausdrucks des Compilers, die ohne Klammern und Vorrang der Operatoren auskommt
- Jeder Operator hat für seine Operanden ein Strich zum Syntaxbaum des Operanden (von links nach rechts)
- Literale, Variablen und Funktionsaufrufe sind Syntaxbäume
- Operator, der zuletzt ausgeführt wird, steht ganz oben im Syntaxbaum

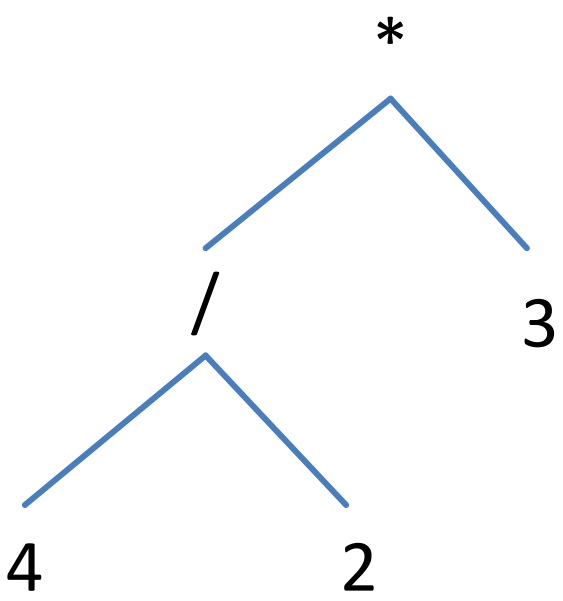
Syntaxbaum

- Beispiele

4 / 2 * 3



4 / (2 * 3)



Syntaxbaum

- Algorithmus, um Syntaxbaum zu erstellen
- Modifikation des shunting-yard Algorithmus von Dijkstra ohne Stapelspeicher

1. Alle Variablen und Literale von links nach rechts hinschreiben. Nach oben Platz lassen.
2. Solange noch ein nicht betrachteter Operator existiert:
   - Von links nach rechts den (ersten, noch nicht betrachteten) Operator mit der höchsten Bindung suchen.
     - Diesen Operator oberhalb der beiden zugehörigen Teilausdrücke im bisherigen Syntaxbaum schreiben
     - Verbindung zum obersten Operator des Syntaxbaums der Teilausdrücke zeichnen (oder zur Variable bzw Literal, falls diese noch nicht Teil eines Syntaxbaums sind).
     - Bei geklammerten Teilausdruck zuerst den zugehörigen Syntaxbaum erstellen.

Ausdrücke / Syntaxbaum

Ausdruck: $(a + b)^2$

$(a + b)^3$ (schwerer)

double a, b;

1. Ausdruck für ausmultiplizierten Term angeben.
   Genau passende Literale verwenden.
2. Entsprechend Vorrang der Operatoren Rechnung klammern.
3. Restliche Teilausdrücke gemäß Linksassoziativität klammern.
4. Syntaxbaum für Ausdruck ohne Klammern aufstellen

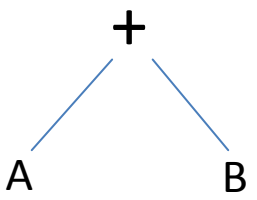
Syntaxbaum (schematisch)

- (A)

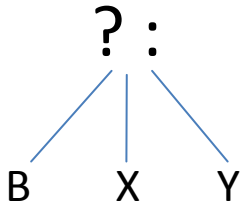
A--

Nicht mehr für Menschen erkennbar,
ob Post- /Präfixoperator

(A) + (B)



B ? X : Y



Konvertierung

- Widening Conversion
  - Automatisch
  - Von niederwertigen Zahltyp zu höherwertigen
  - In wenigen Fällen mit Genauigkeitsverlust
- Narrowing Conversion
  - Cast-Operator nötig (Präfix)
  - Von höherwertigen Zahltyp zu niederwertigen
  - Meist mit Genauigkeitsverlust, oft mit Verlust der Grössenordnung

Narrowing Conversion

- Von höherwertigen Datentyp zu niederwertigen
- Nur mit **Cast-Operator**
  - Unär, Postfix
  - Zieldatentyp wird in ( ) vor Ausdruck geschrieben
- Konvertierung zur Laufzeit oder ggf. Compilezeit
- Cast-Operator auch bei widening conversion verwendbar, aber redundant

```
int i = (int) 2.0;
byte b = (byte) (i + (int) 2.0 );
double d = (double) i; // widening, cast weglassen
```

Narrowing Conversion

- Grobe Konvertierung von Gleitkomma- zu ganzzahligen Datentypen (1. Schritt)
  - Ist der Zieldatentyp eine ganze Zahl und der Ausdruck eine Gleitkommazahl, dann werden die Nachkommastellen weggelassen.
  - Der Vorkommaanteil wird in einen long-Wert umgewandelt, falls Zieldatentyp long, sonst in int
- Weitere Konvertierung (2. Schritt), gilt auch von z.B. long zu int
  - Falls das Resultat zu groß (klein) für den long bzw. int ist, dann wird der maximale (minimale) long bwz. int Wert genommen.
  - Die Repräsentation des Ergebnis wird entsprechend der Größe des Zieldatentyps gekürzt, d.h. die höherwertigen Bits werden einfach verworfen.
- **Narrowing Conversion vermeiden!**

Konvertierung char

- **Spezialfall**
  1. char kann in short, byte mit cast umgewandelt werden
  2. ohne cast zu int, long, float, double
- 16-Bits (8-Bit) des char-Wertes werden als 2er-Komplement interpretiert
  1. Überhängende höherwertige Bits werden verworfen
  2. Fehlende höherwertige Bits werden mit 0en aufgefüllt

```
short s1 = (short) '\u000F';  // 15
short s2 = (short) '\uFFFF';  // -1
byte b1 = (byte) '\u00ff'; // -1
byte b2 = (byte) '1';  // 49
```

Operatoren

- Arithmetische Operatoren
  +, -, *, /, %
  ~, &, |, ^ (Bitweise Operatoren)
- Vergleichsoperatoren
  ==, != (Identitätsoperatoren)
  <, <=, >=, >
- Boolesche Operatoren
  &, |, ^
  &&, || (Kurzschlußoperatoren)
- Zuweisungsoperatoren
  =, ++, --, +=, -=, ...

Arithmetische Operatoren

+ (Addition)

- (Subtraktion)

* (Multiplikation)

/ (Division)

% (Modulo)

- Existieren für alle 6 Zahltypen
- + auch für String

Bitweise Operatoren (1/3)

- Die Operanden haben ganzzahlige Datentypen (**nur** int und long))
- Diese Operatoren operieren auf den Bits der Zahlencodierung
- Die Codierung der Operanden wird nicht als ganze Zahl interpretiert

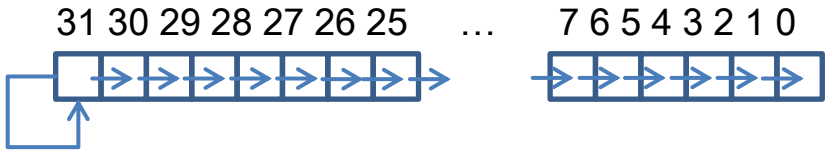
Operator	Beschreibung	
~A	Bits von a werden invertiert 0 wird 1 und 1 wird 0.	~ 0x8000_0006 0x7FFF_FFF9
A & B	Bits von a und b werden stellenweise logisch mit UND verknüpft Wenn beide Bits 1 sind, dann ist das Ergebnisbit 1	0x8000_0006 & 0xF000_0003 0x8000_0002
A ^ B	Verknüpfung mit Exklusivem ODER Wenn beide Bits verschieden sind, ist das Ergebnisbit 1	0x8000_0006 ^ 0xF000_0003 0x7000_0005
A | B	Verknüpfung mit ODER Wenn eines der Bits 1 ist, dann ist das Ergebnisbit 1	0x8000_0006 | 0xF000_0003 0xF000_0007

Bitweise Operatoren (2/3)

- **Nur** für int und long
- a << n : Bits in a werden um n & 0x1F (n & 0x3F, falls a long) Stellen nach links verschoben. Es werden Nullen von rechts nachgeschoben.

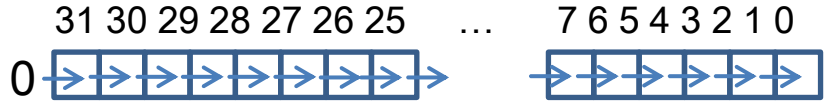
a = 0b1010
a << 2 ist 0000 … 0000 0010 1000

- a >> n : Bits in a werden um n & 0x1F (n & 0x3F, falls a long) Stellen nach rechts verschoben. Das Vorzeichenbit wird nachgeschoben.



a = 0xF00000F
a >> 2 ist 1111 1100 … 000 0011

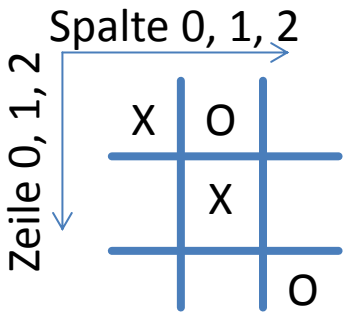
- a >>> n : Wie >> aber es werden Nullen nachgeschoben.

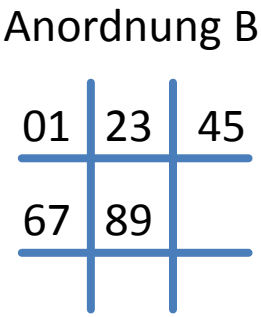


a = 0xF00000F
a >>> 2 ist 0011 1100 … 000 0011

Bitweise Operatoren (3/3)

- Aufgabe
  - Spielfeld Tic-Tac-Toe in einer int-Variablen codieren
  - 9 x 2 Bit nötig
- Codierung für Leeres Spielfeld, X, O mit 2 Bit festlegen, z.B. 00, 10, 01 (binär)
- Geeignete Reihenfolge definieren





010000 001000 000110

Gegeben:	Codiertes Spielfeld, Zeile und Spalte
Gesucht:	Wert des zugehörigen Feldes

- Ein Ausdruck gesucht!

```
int zeile;
int spalte;
int spielfeld;
```

Vergleichsoperatoren

< (kleiner)
> (größer)
<= (kleiner oder gleich)
>= (größer oder gleich)

- Binäre Operatoren
- Nur für Zahltypen
- Ergebnisdatentyp boolean

```
int a = 5;
1 < a       ist true
a > 7       ist false
```

```
1 < a < 8  ist semantisch inkorrekt, da
            die Datentypen der Operanden
             nicht passen:
            (1 < a) < 8     links boolean, rechts int
 (1 < a) & (a < 8)  ist true
 a <= a     ist true
```

Vergleichsoperatoren

Identitätsoperatoren (für alle Datentypen)

- ==, != (binär)
- Diese Operatoren vergleichen nur die Bits der Codierung des linken und rechten Operanden.
- A == B ist true, genau dann, wenn die Bits der Codierung der von A und B identisch sind
- A != B ist true, genau dann, wenn die Codierung verschieden ist
- Ausnahme: -0.0 +0.0
- Bei Datentypen mit eindeutiger Codierung ist "==" die Gleichheit: byte, short, int, long, char, boolean, **sonst nicht**
- Vor allem bei Gleitkommazahlen aufgrund der Rechenungenauigkeiten keine Gleichheit

Vergleichsoperatoren

- Gleitkommazahlen nie direkt mit Identitätsoperatoren vergleichen
- Immer den Abstand zwei Gleitkommazahlen a und b vergleichen

```
Math.abs(a – b) < 0.00000001
```

- Dieser Ausdruck wird true, wenn a und b fast gleich sind
- Problem
  - Wert 0.0000001 hängt vom Wertebereich von a und b
  - Dieser muss für jeden Anwendung individuell gewählt werden

Boolesche Operatoren

- Definition über Verknüpfungstabelle
- Werte linker Operand 1. Spalte,
- Werte rechter Operand in 1. Zeile der anderen Spalten
- Verknüpfungstabelle && und || wie & bzw. |

A & B	true	false
true	true	false
false	false	false

A | B	true	false
true	true	true
false	true	false

A ^ B	true	False
true	false	true
false	true	false

! A	true	false
	false	true

Boolesche Operatoren

- Mit Booleschen Operatoren können logische Aussagen beschrieben werden.
- Elementare Aussagen, wie "es regnet", können mit Booleschen Variablen definiert werden.
- Boolesche Ausdrücke werden meist für Entscheidungen in Kontrollanweisungen verwendet.

Frage:	**Geht Alice auf Büffeljagd?**
Antwort (**Boolesche Ausdruck**)	Alice geht auf Büffeljagd, wenn **sie hungrig und falls sie keine Vegetarierin ist**.

Boolesche Operatoren

- Kurzschlussoperatoren && und ||
- A && B

  B wird nur ausgewertet, wenn A true ist
- A || B

  B wird nur ausgewertet, wenn A false ist
- Kann schneller sein als & und |
- Fehleranfällig, wenn B immer ausgeführt werden soll
- **Nur verwenden, wenn mit A ein Fehlerfall abgefangen werden soll**
- **Als Anfänger vermeiden**

Boolesche Ausdrücke

Gegeben: (Frage):	Ist die Straße nass?
Gesucht (Antwort):	Die Straße ist nass, wenn **ein Hochwasserdamm bricht und der angestaute See nicht leer ist** **oder** **es regnet**.

1. Variablem mit geeigneten Datentypen für elementare Aussagen deklarieren
2. Booleschen Ausdruck für fett gedruckten Sachverhalt angeben

Zuweisungsoperatoren

- Existiert für alle Datentypen
- Syntax:
  Variable = Ausdruck;
- Variable und Ausdruck müssen den gleichen Datentyp haben.
- Der Wert der Variable ändert sich zum Wert des Ausdrucks.
- Das Ergebnis des Zuweisungsoperators ist der Wert des Ausdrucks
- Die Zuweisung kann als Anweisung verwendet werden.
- Seiteneffekt (in einem Ausdruck): Werte von Variablen ändern sich (während Ausführung des Ausdrucks)
- **Seiteneffekte in Ausdrücken vermeiden!**

```
int a = 1;
int b;
b = ((a = 2) + 1);
// Ergebnis von a = 2 ist 2, a ist 2, b ist 3
```

Zuweisungsoperatoren

- Syntax
  Variable Op= Ausdruck
  wobei Op ein Operator ist
- Dies Ist eine Abkürzung für
  Variable = (Variable) Op Ausdruck
- **Verwenden, wenn Programm dadurch lesbarer wird**
- **Als Anfänger meiden**

```
int a = 2;

a += 2; // a wird um 2 erhöht
// a ist 4
int a = 3;
int b = (a = 2) * a;
// b ist 4
```

Zuweisungsoperatoren

- Präinkrement / -dekrement
- Unär, für alle sechs Zahltypen

  ++Variable <=> Variable += 1

  --Variable <=> Variable -= 1
- Postinkrement / -dekrement
- Ergebnis ist Wert der Variable *vor* Änderung

  Variable++ Die Variable wird um Eins erhöht

  Variable-- Variable wird um Eins reduziert

```
int a = 2;
int b;

b = (a++);     // b ist 2, a ist 3
b = (--a);     // b ist 2, a ist 2
```

Zuweisung mit Seiteneffekten

Aufgabe

```
int a = 1;

a = ((a = 2) + (a = (a + 1)));
```

Syntaxbaum erstellen
Werten Sie den Ausdruck Schritt-für-Schritt aus und notieren sie die Zwischenergebnisse der Teilausdrücke

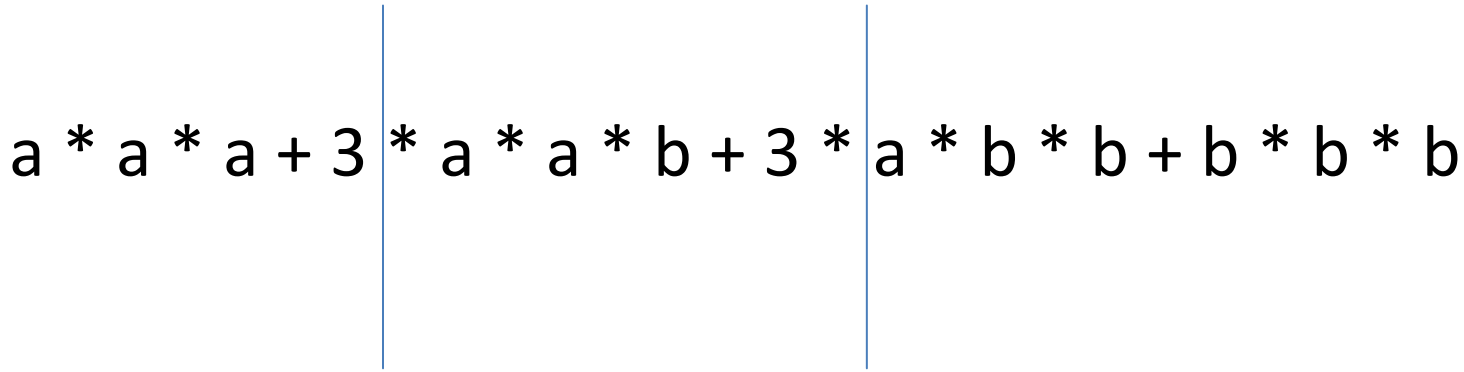
Konvention Ausdrücke (1/2)

Konvention	Beispiele
Keine Variablenwerte in Ausdrücken ändern (Seiteneffekte vermeiden)	
Insbesondere keine Zuweisungsoperatoren in Ausdrücken verwenden	int a = 0; a = (a += (a = a + 1))

Konvention Ausdrücke (2/2)

Konvention	Beispiele
Vor und nach einem binären Operator ein Leerzeichen setzen.	1 * 2 * langerName / 8695784.0
Überlange Ausdrücke vor dem Operator mit schwächster Bindung umbrechen. Klammerung dabei berücksichtigen.	(a * b * c) + (d * e * f) + (g + h + i)
Teilausdruck Einrücken zum Anfang des zugehörigen linken Operators (falls möglich)	wert1 * wert2 / wert3 – wert5 * wert6 / wert7

Ausdruck formatieren



Ausdruck jeweils an beiden „Bildschirm“-Markierungen umbrechen