3. Ganze Zahlen

Auswertung arithmetischer Ausdrücke, Assoziativität und Präzedenz, arithmetische Operatoren, Wertebereich der Typen `int`, `unsigned int`

Celsius to Fahrenheit

```
// Program: fahrenheit.cpp
// Convert temperatures from Celsius to Fahrenheit.
#include <iostream>

int main() {
  // Input
  std::cout << "Temperature in degrees Celsius =? ";
  int celsius;
  std::cin >> celsius;

  // Computation and output
  std::cout << celsius << " degrees Celsius are "
            << 9 * celsius / 5 + 32  << " degrees Fahrenheit.\n";
  return 0;
}
```

Celsius to Fahrenheit

```
// Program: fahrenheit.cpp
// Convert temperatures from Celsius to Fahrenheit.
#include <iostream>

int main() {
  // Input
  std::cout << "Temperature in degrees Celsius =? ";
  int celsius;
  std::cin >> celsius;

  // Computation and output
  std::cout << celsius << " degrees Celsius are "
            << 9 * celsius / 5 + 32  << " degrees Fahrenheit.\n";
  return 0;
}
```

`9 * celsius / 5 + 32`

- Arithmetischer Ausdruck,

```
9 * celsius / 5 + 32
```

- Arithmetischer Ausdruck,
- drei Literale, eine Variable, drei Operatorsymbole

```
9 * celsius / 5 + 32
```

- Arithmetischer Ausdruck,
- drei Literale, eine Variable, drei Operatorsymbole

```
9 * celsius / 5 + 32
```

- Arithmetischer Ausdruck,
- drei Literale, eine Variable, drei Operatorsymbole

`9 * celsius / 5 + 32`

- Arithmetischer Ausdruck,
- drei Literale, eine Variable, drei Operatorsymbole

Wie ist der Ausdruck geklammert?

Präzedenz

Punkt vor Strichrechnung

```
9 * celsius / 5 + 32
```

bedeutet

```
(9 * celsius / 5) + 32
```

Präzedenz

Regel 1: Präzedenz

Multiplikative Operatoren (`*`, `/`, `%`) haben höhere Präzedenz ("binden stärker") als additive Operatoren (`+`, `-`)

Assoziativität

Von links nach rechts

```
9 * celsius / 5 + 32
```

bedeutet

```
((9 * celsius) / 5) + 32
```

Assoziativität

Regel 2: Assoziativität

Arithmetische Operatoren (`*`, `/`, `%`, `+`, `-`) sind linksassoziativ: bei gleicher Präzedenz erfolgt Auswertung von links nach rechts

Stelligkeit

Regel 3: Stelligkeit

Unäre Operatoren +, − vor binären +, −.

```
-3 - 4
```

bedeutet

```
(-3) - 4
```

Klammerung

Jeder Ausdruck kann mit Hilfe der

- Assoziativitäten
- Präzedenzen
- Stelligkeiten

der beteiligten Operatoren eindeutig geklammert werden.

Ausdrucksbäume

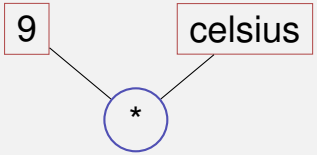
Klammerung ergibt Ausdrucksbaum

```
9 * celsius / 5 + 32
```

Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

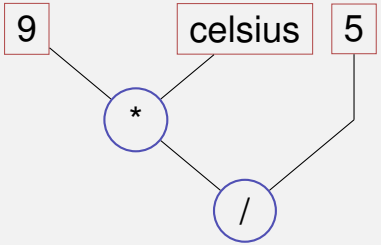
```
(9 * celsius) / 5 + 32
```



Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

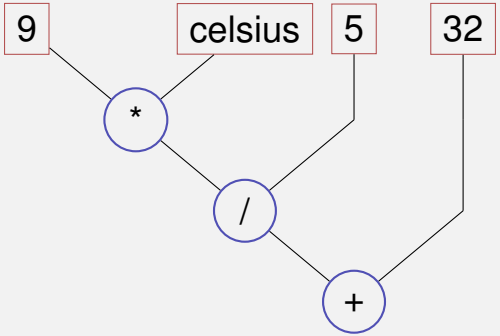
```
((9 * celsius) / 5) + 32
```



Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

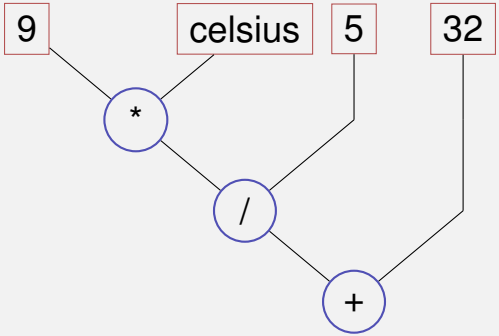
```
(((9 * celsius) / 5) + 32)
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

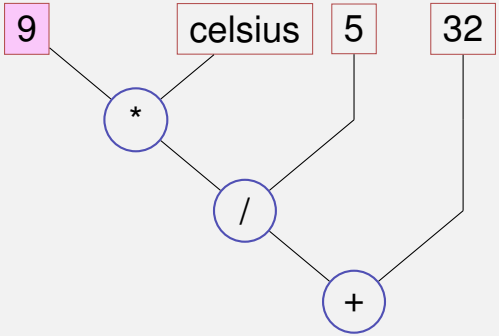
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

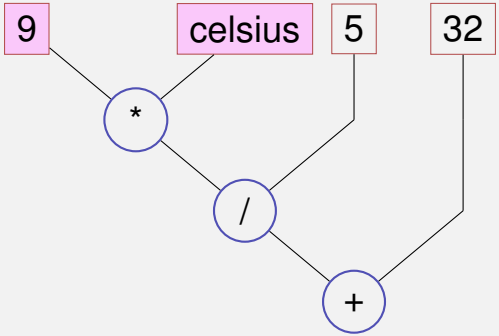
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

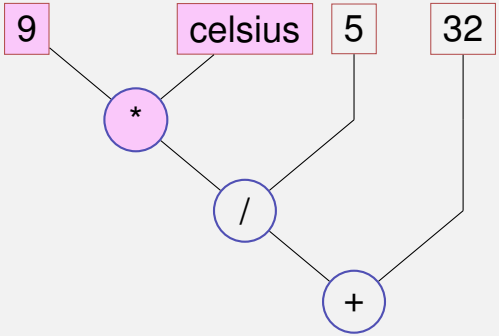
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

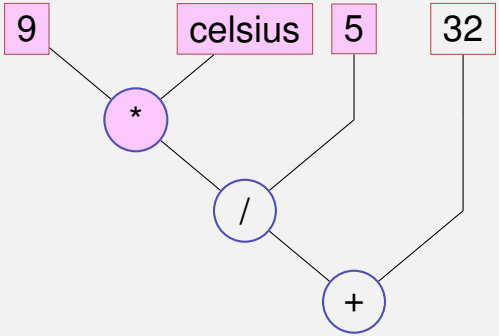
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

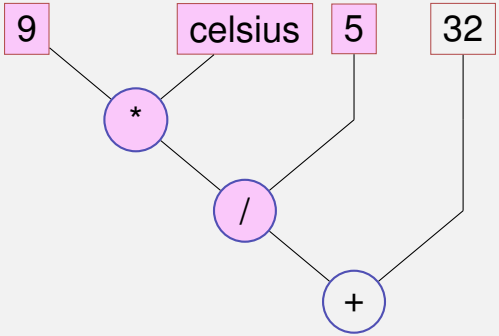
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

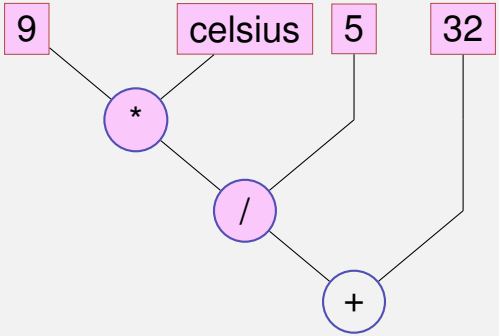
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

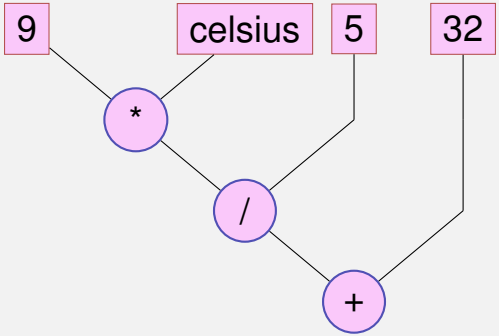
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

"Von oben nach unten" im Ausdrucksbaum

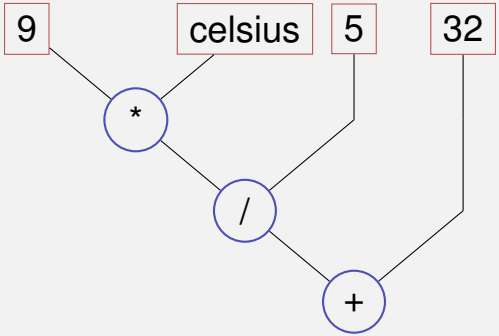
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

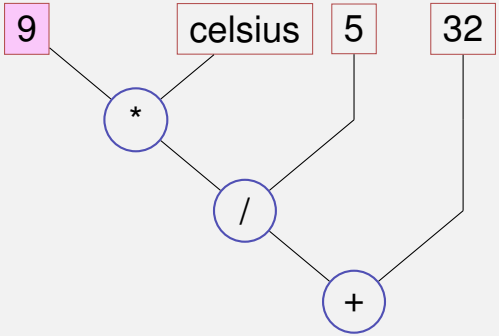
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

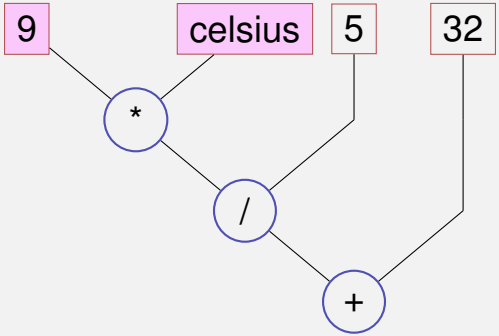
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

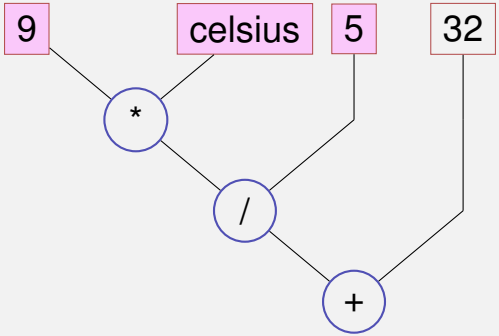
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

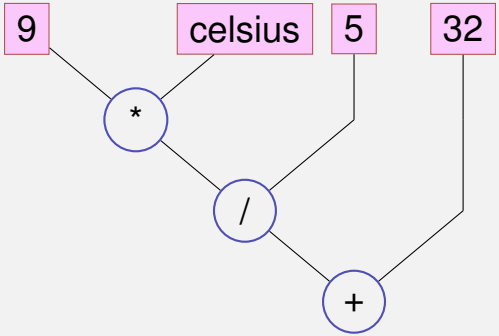
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

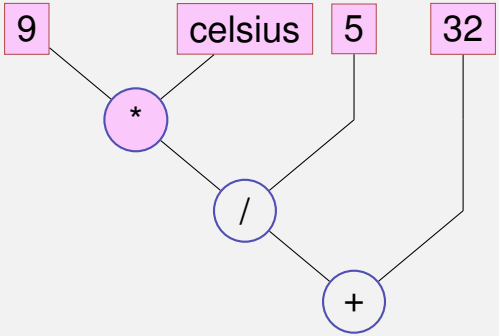
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

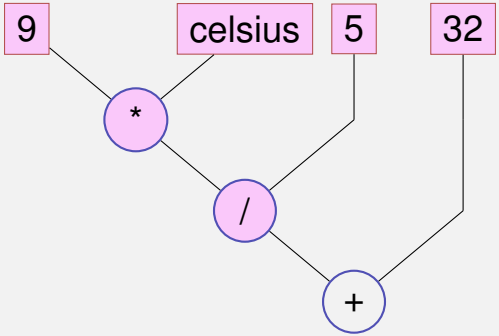
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

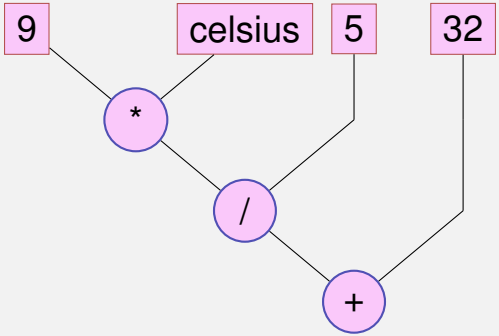
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

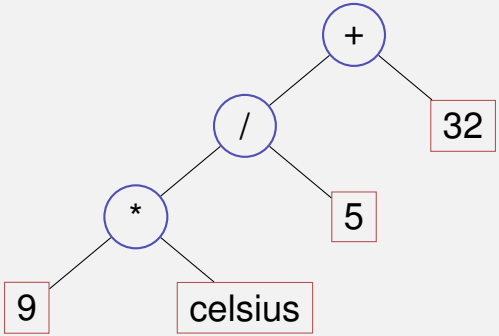
```
9 * celsius / 5 + 32
```



Ausdrucksbäume – Notation

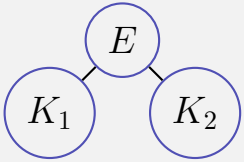
Üblichere Notation: Wurzel oben

```
9 * celsius / 5 + 32
```



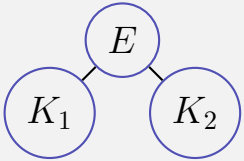
Auswertungsreihenfolge – formaler

- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



Auswertungsreihenfolge – formaler

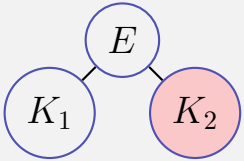
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

Auswertungsreihenfolge – formaler

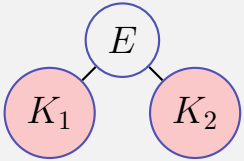
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

Auswertungsreihenfolge – formaler

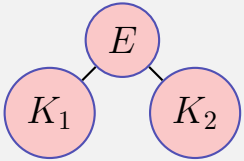
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

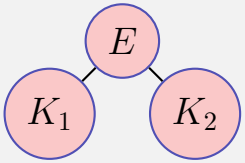
Auswertungsreihenfolge – formaler

- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

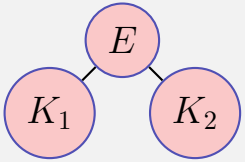
Auswertungsreihenfolge – formaler

- 

C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

- "Guter Ausdruck": jede gültige Reihenfolge führt zum gleichen Ergebnis.

Auswertungsreihenfolge – formaler

- 

C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

- Beispiel für "schlechten Ausdruck": `(a+b)*(a++)`

Auswertungsreihenfolge

Richtlinie

Vermeide das Verändern von Variablen, welche im selben Ausdruck noch einmal verwendet werden!

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	**-**	**1**	**16**	**rechts**
Multiplikation	*	2	14	links
Division	/	2	14	links
Modulus	%	2	14	links
Addition	+	2	13	links
Subtraktion	-	2	13	links

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	**−**	**1**	**16**	**rechts**
Multiplikation	∗	2	14	links
Division	/	2	14	links
Modulus	%	2	14	links
Addition	+	2	13	links
Subtraktion	−	2	13	links

-a : R-Wert $\rightarrow$ R-Wert

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	−	1	16	rechts
Multiplikation	*	2	14	links
Division	/	2	14	links
Modulus	%	2	14	links
Addition	**+**	**2**	**13**	**links**
Subtraktion	−	2	13	links

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	−	1	16	rechts
Multiplikation	∗	2	14	links
Division	/	2	14	links
Modulus	%	2	14	links
Addition	**+**	**2**	**13**	**links**
Subtraktion	−	2	13	links

$$a+b : \text{R-Wert} \times \text{R-Wert} \rightarrow \text{R-Wert}$$

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	−	1	16	rechts
Multiplikation	*	2	14	links
Division	/	2	14	links
Addition	+	2	13	links
Subtraktion	−	2	13	links

```
-a+b+c
```

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	−	1	16	rechts
Addition	+	2	13	links
Subtraktion	−	2	13	links

```
-a+b+c
```

Arithmetische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Unäres +	+	1	16	rechts
Negation	−	**1**	**16**	**rechts**
Addition	+	**2**	**13**	**links**
Subtraktion	−	2	13	links

-a+b+c = ((-a) + b) + c

R-Wert $\times$ R-Wert $\times$ R-Wert $\rightarrow$ R-Wert

Zuweisungsausdruck – nun genauer

- Bereits bekannt: `a = b` bedeutet
  Zuweisung von `b` (R-Wert) an `a` (L-Wert).
  Rückgabe: L-Wert

Zuweisungsausdruck – nun genauer

- Bereits bekannt: `a = b` bedeutet
  Zuweisung von `b` (R-Wert) an `a` (L-Wert).
  Rückgabe: L-Wert
- Was bedeutet `a = b = c` ?

Zuweisungsausdruck – nun genauer

- Bereits bekannt: `a = b` bedeutet
  Zuweisung von `b` (R-Wert) an `a` (L-Wert).
  Rückgabe: L-Wert
- Was bedeutet `a = b = c` ?
- Antwort: Zuweisung rechtsassoziativ, also

```
a = b = c        ⟺        a = (b = c)
```

Zuweisungsausdruck – nun genauer

`a = b = c` $\iff$ `a = (b = c)`

Beispiel Mehrfachzuweisung:

`a = b = 0` $\Longrightarrow$ `b=0; a=0`

Division und Modulus

- Operator / realisiert ganzzahlige Division

  `5 / 2` hat Wert 2

Division und Modulus

- Operator `/` realisiert ganzzahlige Division

  `5 / 2` hat Wert 2

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32

15 degrees Celsius are 59 degrees Fahrenheit
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent...

```
9 / 5 * celsius + 32
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent...

```
1 * celsius + 32
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent...

```
15 + 32
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent...

```
47
```

Division und Modulus

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32
```

```
15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent. . . aber nicht in C++!

```
9 / 5 * celsius + 32
```

```
15 degrees Celsius are  47 degrees Fahrenheit
```

Division und Modulus

- Modulus-Operator berechnet Rest der ganzzahligen Division

`5 / 2` hat Wert 2, `5 % 2` hat Wert 1.

Division und Modulus

- Modulus-Operator berechnet Rest der ganzzahligen Division

  `5 / 2` hat Wert 2, `5 % 2` hat Wert 1.

- Es gilt immer:

  `(a / b) * b + a % b` hat den Wert von `a`.

Inkrement und Dekrement

- Erhöhen / Erniedrigen einer Zahl um 1 ist eine häufige Operation
- geht für einen L-Wert so:

```
expr = expr + 1.
```

Inkrement und Dekrement

```
expr = expr + 1.
```

Nachteile

Inkrement und Dekrement

```
expr = expr + 1.
```

Nachteile

- relativ lang
- expr wird zweimal ausgewertet (Effekte!)

In-/Dekrement Operatoren

Post-Inkrement

```
expr++
```

Wert von expr wird um 1 erhöht, der *alte* Wert von expr wird (als R-Wert) zurückgegeben

In-/Dekrement Operatoren

Prä-Inkrement

```
++expr
```

Wert von expr wird um 1 erhöht, der *neue* Wert von expr wird (als L-Wert) zurückgegeben

In-/Dekrement Operatoren

Post-Dekrement

```
expr--
```

Wert von expr wird um 1 verringert, der *alte* Wert von expr wird (als R-Wert) zurückgegeben

In-/Dekrement Operatoren

Prä-Dekrement

```
--expr
```

Wert von expr wird um 1 verringert, der *neue* Wert von expr wird (als L-Wert) zurückgegeben

In-/Dekrement Operatoren

Beispiel

```
int a = 7;
std::cout << ++a << "\n";
std::cout << a++ << "\n";
std::cout << a << "\n";
```

In-/Dekrement Operatoren

Beispiel

```
int a = 7;
std::cout << ++a << "\n"; // 8
std::cout << a++ << "\n";
std::cout << a << "\n";
```

In-/Dekrement Operatoren

Beispiel

```
int a = 7;
std::cout << ++a << "\n"; // 8
std::cout << a++ << "\n"; // 8
std::cout << a << "\n";
```

In-/Dekrement Operatoren

Beispiel

```
int a = 7;
std::cout << ++a << "\n"; // 8
std::cout << a++ << "\n"; // 8
std::cout << a << "\n"; // 9
```

C++ vs. ++C

Eigentlich sollte unsere Sprache ++C heissen, denn

- sie ist eine Weiterentwicklung der Sprache C,

C++ vs. ++C

Eigentlich sollte unsere Sprache ++C heissen, denn

- sie ist eine Weiterentwicklung der Sprache C,
- während C++ ja immer noch das alte C liefert.

Arithmetische Zuweisungen

```
a += b
```

$\Leftrightarrow$

```
a = a + b
```

Arithmetische Zuweisungen

```
a += b
```

$\Leftrightarrow$

```
a = a + b
```

Analog für `-`, `*`, `/` und `%`

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: `101011`

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: `101011` entspricht `32+8+2+1`.

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: `101011` entspricht `43`.

Binäre Zahlendarstellungen

Binäre Darstellung ("Bits" aus $\{0, 1\}$)

$$b_n b_{n-1} \dots b_1 b_0$$

entspricht der Zahl $b_n \cdot 2^n + \dots + b_1 \cdot 2 + b_0$

Beispiel: 101011 entspricht 43.

Binäre Zahlen: Zahlen der Computer?

Wahrheit: Computer rechnen mit Binärzahlen.

NEUE ZÜRCHER ZEITUNG

TECHNIK

Mittwoch, 30. August 1950 Blatt 5
Mittagausgabe Nr. 2736 (50)

Das programmgesteuerte Rechengerät an der Eidgenössischen Technischen Hochschule in Zürich

Die Entwicklung programmgesteuerter Rechenmaschinen in den Vereinigten Staaten von Amerika wurde in den Artikeln „Elektronische Rechenmaschinen" (vgl. Nr. 2140 der „N. Z. Z." vom 13. Oktober 1948) und „Die neueste elektronische Rechenmaschine" (vgl. Nr. 871 der „N. Z. Z." vom 26. April 1950) behandelt. Nachstehend soll von einem Gerät deutscher Herkunft — Zuse K.-G., Neukirchen — die Rede sein, welches im Juli dieses Jahres am Institut für angewandte Mathematik der Eidgenössischen Technischen Hochschule in Zürich, das unter der Leitung von Prof. Dr. E. Stiefel steht, in Betrieb genommen wurde. Damit ist diesem Institut in der Lage, dem in der Schweiz immer stärker werdenden Bedürfnis nach einer leistungsfähigen Zentralstelle für numerische Rechnungen wenigstens teilweise gerecht zu werden. Bereits sind einige mathematische Probleme behandelt worden, und die Erledigung vieler weiterer Aufgaben ist vorbereitet.

Merkmale des Gerätes

Das Gerät ist ein Glied in dem längeren Entwicklungsprogramm des Ingenieurs *Konrad Zuse*; es wurde im Auftrag des *Institutes für angewandte Mathematik* der E. T. H. unter Berücksichtigung von dessen Wünschen und Ideen von Zuse als *„Modell Z 4"* konstruiert. Die ursprüngliche Entwicklung in Deutschland erfolgte in den Kriegsjahren und verlief völlig unabhängig von den Unternehmungen in den Vereinigten Staaten. Es ist übrigens interessant festzustellen, wie für die meisten wichtigen funktionellen Probleme beiderseits genau dieselbe Lösung gefunden wurde, wie aber anderseits gewissen Fragen sekundärer Wichtigkeit eine ganz unterschiedliche Bedeutung beigemessen wurde.

Eine kurze technische Charakterisierung lautet wie folgt: Elektromechanisch arbeitendes Gerät mit 2200 Relais, 21 Schaltschaltern und einem Speicher für 64 Zahlen, welcher mit neuartigen, mechanischen Schaltgliedern arbeitet; Verwendung des Dualsystems und der halblogarithmischen Darstellung; Multiplikationszeit 2,5 Sekunden; Programmsteuerung mit Hilfe zweier Lochstreifen, auf die wahlweise umgeschaltet werden kann; Eingabe von Zahlen durch eine Tastatur oder durch einen Lochstreifen; Angabe der Resultate durch Lampenfeld, Lochstreifen oder Druckwerk.

Das duale Zahlsystem

Allgemein wird programmgesteuerten Rechengeräten häufig das duale Zahlsystem zugrunde gelegt, welches nur die zwei Zahlzeichen 0 und 1 verwendet, während das bekannte Dezimalsystem

Lesen wir eine Dezimalzahl von rechts nach links, so erhöht sich das Gewicht von Stelle zu Stelle um den Faktor 10. Im *Dualsystem* ist nun einfach dieser Faktor 10 durch 2 zu ersetzen. Also bedeutet die (nunmehr duale) Zahl abcdefg den Ausdruck:

$$a \cdot 2^6 + b \cdot 2^5 + c \cdot 2^4 + d \cdot 2^3 + e \cdot 2^2 + f \cdot 2^1 + g \cdot 2^0$$

Die Zahl 1 wird in beiden Systemen gleich dargestellt. Um jedoch duale von dezimalen Zahlen deutlich zu trennen, schreiben wir die duale 1 als 1_D. — Dagegen weicht schon die 2 ab, indem sie dual 10_D lautet; denn dies bedeutet ja $1 \cdot 2^1 + 0 \cdot 2^0 = 2$. Wenn einer Zahl (ohne Stellen nach dem Komma) rechts eine Null zugefügt wird, so vergrößert sie sich um den Faktor 2 (und nicht, wie im Dezimalsystem, um den Faktor 10). Auf diese Weise kann aus $1_D 0 = 2$ auf einfachste Weise gebildet werden: $1_D 00 = 4$, $1_D 000 = 8$, $1_D 0000 = 16$, usw.

Die Dualzahl *LOLAL* bedeutet nun also:

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 21$$

Ganz analog sind etwaige Stellen nach dem Komma zu interpretieren; so wird z. B. *0,LL* wie folgt übersetzt:

$$1 \cdot 2^{-2} + 1 \cdot 2^{-1} = 1 \cdot 2^{-2} + 1 \cdot 2^{-1} = 1 \cdot {}^1/_4 + 1 \cdot {}^1/_2 = {}^3/_4 = 1{,}375$$

Der große Vorteil, der das Dualsystem für Rechenmaschinen so geeignet macht, nämlich die Reduktion der Anzahl der verwendeten Symbole auf nur zwei, wird durch den Nachteil eines Nachteil erkauft: Es braucht mehr Stellen, um eine bestimmte Zahl darzustellen. Die einstellige Zahl 8

Änderung des Maßstabes durchgerechnet werden können.

Die beschriebene Darstellung bringt eine gewisse Komplikation der Rechenoperationen mit sich. So müssen vor einer Addition die beiden Summanden zunächst so verschoben werden, daß ihre Kommata untereinander zu liegen kommen, was an Hand eines Beispiels erläutert werden soll. Damit der Leser nicht durch das ungewohnte duale Zahlsystem verwirrt wird, ist das Beispiel im Dezimalsystem durchgeführt; doch wird darum ersetzt, daß das Gerät in Wirklichkeit mit dualen Zahlen rechnet.

Es soll also etwa addiert werden: $2{,}345678 \times 10^5 + 9{,}876543 \times 10^{-1}$ (Man beachte, daß die eigentliche Zahl stets zwischen 1 und 10 liegt, also das Komma nach der ersten Stelle hat.) Nun müssen die beiden Summanden „ausgerichtet" werden, d. h. die beiden Exponenten sind einander gleich zu machen, und zwar erhält der kleinere Exponent den Wert des größeren, also 2. Die Zahlen lauten nun, richtig untereinander geschrieben und addiert, wie folgt:

$$\begin{array}{l} 2{,}345678 \times 10^5 \\ 0{,}009876 \times 10^5 \\ \hline 2{,}355554 \times 10^5 \end{array}$$

Es ist ersichtlich, daß bei der kleineren der beiden Zahlen rechts einige Stellen abgeschnitten werden müssen; dann wenn die Summanden siebenstellig angegeben waren, so soll auch das Resultat nicht mehr als sieben Stellen enthalten.

Abb. 2. Der Schaltpult bei der Fertigung eines Rechenplanes. Die Abtaster für den Lochstreifen sind deutlich sichtbar.

Befehle können „bedingt" gegeben werden, d. h. ihre Ausführung wird von der Natur eines errechneten Resultates abhängig gemacht. Erst dadurch werden die mathematischen Probleme

Binäre Zahlen: Zahlen der Computer?

Klischee: Computer reden 0/1-Kauderwelsch.

Überall Windmühlen – Gustl Mollath und die Gerechtigkeit › Die Seite Drei

Süddeutsche Zeitung

NEUESTE NACHRICHTEN AUS POLITIK, KULTUR, WIRTSCHAFT UND SPORT

WWW.SÜDDEUTSCHE.DE | HH2 | MÜNCHEN, SAMSTAG/SONNTAG, 26./27. JULI 2014 | 70. JAHRGANG / 30. WOCHE / NR. 170 / 2,50 EURO

Das Streiflicht

(SZ) Carl Barks, der Erfinder der nach wie vor unumstritten reichsten Ente der Welt, war für viele ein großer Künstler. Für viele andere war der Comicautor und Maler nichts weiter als ein harmloser Entenzeichner. Die im Land der Dichter und Denker schmerzhaft genaue Unterscheidung zwischen Hoch- und Niedrigkultur wird auch bei diesem großen Humoristen unbarmherzig durchgezogen. Amerika macht sich in Sachen Popkultur lockerer. Dort gelten Barks' Geschichten als Meisterstücke der gesellschaftskritischen Satire: Wenn Dagobert Duck Gier und Geiz zu immer neuen Höhen trieb, stand plötzlich auch der amerikanische Turbokapitalismus mit einem aufgeregt wackelnden Entensterz da. Hierzulande sieht man in Barks immer noch eher den Lieferanten kindgerechter Zerstreuung. Allein musste er sich dabei nicht fühlen, gegen dieses Urteil kämpft jeder Comic-Künstler fortwährend an.

Nun haben Barks, alle Comiczeichner und überhaupt alle Menschen, die in einer Gastwirtschaft der Nachwelt auf Bierdeckeln lustig anzuschauende Figürchen hinterlassen, himmlischen Beistand erhalten. Auch das Universum zeichnet Enten. Und nicht die schlechtesten, um das gleich hinzuzufügen. Die Menschheit muss nur mal genau hinschauen beziehungsweise eine russische Raumsonde losschicken, und schon können wir erkennen, dass der Komet 67P/Tschurjumov-Gerasimenko liebevoll kurz „Tschuri" genannt, in Form einer Ente durchs Weltall saust. Das Universum kann auch anders, mit diesem Stoßseufzer dürfen wir fortan zum Himmel blicken. Kometen, das sind nicht zwangsläufig Angst und Schrecken verbreitende Himmelszeichen, die mit loderndem Schweif Pest,

Der Geheimcode

Sie stecken im Fahrkartenautomaten, in der Zentralheizung, im Handy: kleine Computer. Und sie tun nur deshalb, was sie tun sollen, weil sie programmiert sind – in Programmiersprachen. Aber was sind das für Zeichen, die mittlerweile unser Leben bestimmen? Höchste Zeit, sie zu verstehen

› Feuilleton

ILLUSTRATION: STEFAN DIMITROV/SZ

WOCHENENDE

Liebe Nur Patriarchen und Kapuzenpullischluffis unterwegs? Wir Männer sind weiter, als wir glauben. JAKOB SCHRENK

Luxus Während der Finanzkrise verschwand der Kaviar von der Speisekarte. Zu Besuch bei Züchtern, Händlern und Kunden. TIM NESHITOV

Punk Berlins Kulturstaatssekretär Tim Renner im Interview. MARTIN ZIPS

Mit Stellenmarkt

Anklage wegen Verwandtenaffäre

Ex-CSU-Fraktionschef Schmid soll sich vor Gericht verantworten

Augsburg – Die Staatsanwaltschaft Augsburg hat Anklage gegen den ehemaligen CSU-Fraktionschef Georg Schmid erhoben. Der 61-Jährige soll mindestens 340 000 Euro Sozialversicherungsbeiträge hinterzogen haben. Nach Angaben

Binäre Zahlen: Zahlen der Computer?

Klischee: Computer reden 0/1-Kauderwelsch.

01001110 01011010 01011010

Freitag, 8. Juni 2012 · Nr. 131 · 233. Jhg. *01001010 01010110 01001101* www.nzz.ch · Fr. 4.00 · €3.50

01000010 01100101
01110010 01101001

01100011 01101000 01110100 01100101

01110011 01100001

01100110 01101100 11111100

01101000

Rechentricks

- Abschätzung der Grössenordnung von Zweierpotenzen[3]:

$2^{10} = 1024 = 1\text{Ki} \approx 10^3$.
$2^{32} = 4 \cdot (1024)^3 = 4\text{Gi}$.
$2^{64} = 16\text{Ei} \approx 16 \cdot 10^{18}$.

[3] Dezimal vs. Binäre Einheiten: MB - Megabyte vs. MiB - Megabibyte (etc.)
kilo (K, Ki) – mega (M, Mi) – giga (G, Gi) – tera(T, Ti) – peta(P, Pi) – exa (E, Ei)

Rechentricks

- Abschätzung der Grössenordnung von Zweierpotenzen[3]:

$2^{10} = 1024 = 1\text{Ki} \approx 10^3.$
$2^{32} = 4 \cdot (1024)^3 = 4\text{Gi}.$
$2^{64} = 16\text{Ei} \approx 16 \cdot 10^{18}.$

[3] Dezimal vs. Binäre Einheiten: MB - Megabyte vs. MiB - Megabibyte (etc.)
kilo (K, Ki) – mega (M, Mi) – giga (G, Gi) – tera(T, Ti) – peta(P, Pi) – exa (E, Ei)

Rechentricks

- Abschätzung der Grössenordnung von Zweierpotenzen[3]:

$2^{10} = 1024 = 1\text{Ki} \approx 10^3.$
$2^{32} = 4 \cdot (1024)^3 = 4\text{Gi}.$
$2^{64} = 16\text{Ei} \approx 16 \cdot 10^{18}.$

[3] Dezimal vs. Binäre Einheiten: MB - Megabyte vs. MiB - Megabibyte (etc.)
kilo (K, Ki) – mega (M, Mi) – giga (G, Gi) – tera(T, Ti) – peta(P, Pi) – exa (E, Ei)

Hexadezimale Zahlen

Zahlen zur Basis 16. Darstellung

$$h_n h_{n-1} \dots h_1 h_0$$

entspricht der Zahl

$$h_n \cdot 16^n + \dots + h_1 \cdot 16 + h_0.$$

Schreibweise in C++: vorangestelltes `0x`

Beispiel: `0xff` entspricht `255`.

Hex Nibbles

hex	bin	dec
0	0000	0
1	**0001**	**1**
2	**0010**	**2**
3	0011	3
4	**0100**	**4**
5	0101	5
6	0110	6
7	0111	7
8	**1000**	**8**
9	1001	9
a	1010	10
b	1011	11
c	1100	12
d	1101	13
e	1110	14
f	1111	15

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.
- „Kompakte Darstellung von Binärzahlen”.

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.
- „Kompakte Darstellung von Binärzahlen”.

32-bit Zahlen: `0x00000000` -- `0xffffffff` .
`0x80000000`: höchstes Bit einer 32-bit Zahl gesetzt.
`0xffffffff`: alle Bits einer 32-bit Zahl gesetzt.
„`0x8a20aaf0` ist eine Addresse in den oberen 2G des 32-bit Addressraumes”

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.
- „Kompakte Darstellung von Binärzahlen”.

32-bit Zahlen: `0x00000000` -- `0xffffffff` .
`0x80000000`: höchstes Bit einer 32-bit Zahl gesetzt.
`0xffffffff`: alle Bits einer 32-bit Zahl gesetzt.
„`0x8a20aaf0` ist eine Addresse in den oberen 2G des 32-bit Addressraumes”

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.
- „Kompakte Darstellung von Binärzahlen”.

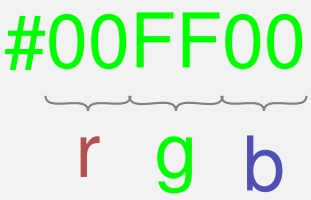
32-bit Zahlen: `0x00000000` -- `0xffffffff` .
`0x80000000`: höchstes Bit einer 32-bit Zahl gesetzt.
`0xffffffff`: alle Bits einer 32-bit Zahl gesetzt.
„`0x8a20aaf0` ist eine Addresse in den oberen 2G des 32-bit Addressraumes”

Wozu Hexadezimalzahlen?

- Ein Hex-Nibble entspricht *genau* 4 Bits.
- „Kompakte Darstellung von Binärzahlen”.

32-bit Zahlen: `0x00000000` -- `0xffffffff` .
`0x80000000`: höchstes Bit einer 32-bit Zahl gesetzt.
`0xffffffff`: alle Bits einer 32-bit Zahl gesetzt.
„`0x8a20aaf0` ist eine Addresse in den oberen 2G des 32-bit Addressraumes”

Beispiel: Hex-Farben



Beispiel: Hex-Farben

Beispiel: Hex-Farben

Beispiel: Hex-Farben

Wozu Hexadezimalzahlen?

“Für Programmierer und Techniker”

(Bedienungsanleitung Schachcomputer *Mephisto II*, 1981)

Beispiele:

8200

a) Anzeige ***8200***
MEPHISTO ist mit genau 2 Bauern-Einheiten im Vorteil.

7F00

b) Anzeige ***7F00***
MEPHISTO ist mit genau 1 Bauern-Einheit im Nachteil.

Die Anzeige erfolgt in ***hexadezimaler Schreibweise.*** Im Gegensatz zum gewohnten Dezimalsystem gehen die Ziffern an jeder Stelle von 0 bis F (A = 10, B = 11, ... F = 15).
Für mathematisch Vorgebildete nachstehend die Umrechnungsformel in das dezimale Punktsystem:

$$ABCD = (Ax16^3) + (Bx16^2) + (Cx16^1) + (Dx16^0)$$

Für A gilt: 7 = -1; 8 = 0; 9 = +1 usw.
Eine Bauerneinheit (B) wird ausgedrückt in $16^2 = 256$ Punkten.
Dieses auf den ersten Blick vielleicht etwas komplizierte System dient der Service-Freundlichkeit von MEPHISTO, sowie insbesondere der Entwicklungsarbeit an zukünftigen, noch stärkeren Programmen, ist also mehr für unsere Programmierer und Techniker vorgesehen.

Beispiele:

805E

c) Anzeige ***805E***
(E=14) Umrechnung nach folgendem Verfahren:
$(14x16^0) + (5x16^1) + (0x16^2) + (0x16^3) = 14+80+0+0 =$
$= +94$ ***Punkte.***

7F80

d) Anzeige ***7F80***
(7=-1; F=15) Umrechnung wie folgt:
$(0x16^0) + (8x16^1) + (15x16^2) - (1x16^3) = 0+128+3840-4096 =$

Wozu Hexadezimalzahlen?

Die NZZ hätte viel Platz sparen können...

Wertebereich des Typs `int`

```
// Program: limits.cpp
// Output the smallest and the largest value of type int.

#include <iostream>
#include <limits>

int main()
{
  std::cout << "Minimum int value is "
            << std::numeric_limits<int>::min() << ".\n"
            << "Maximum int value is "
            << std::numeric_limits<int>::max() << ".\n";
  return 0;
}
```

Wertebereich des Typs `int`

```
// Program: limits.cpp
// Output the smallest and the largest value of type int.

#include <iostream>
#include <limits>

int main()
{
  std::cout << "Minimum int value is "
            << std::numeric_limits<int>::min() << ".\n"
            << "Maximum int value is "
            << std::numeric_limits<int>::max() << ".\n";
  return 0;
}
```

Zum Beispiel

```
Minimum int value is -2147483648.
Maximum int value is 2147483647.
```

Wertebereich des Typs `int`

```
// Program: limits.cpp
// Output the smallest and the largest value of type int.

#include <iostream>
#include <limits>

int main()
{
  std::cout << "Minimum int value is "
            << std::numeric_limits<int>::min() << ".\n"
            << "Maximum int value is "
            << std::numeric_limits<int>::max() << ".\n";
  return 0;
}
```

Zum Beispiel

```
Minimum int value is -2147483648.
Maximum int value is 2147483647.
```

Woher kommen diese Zahlen?

Wertebereich des Typs `int`

- Repräsentation mit B Bits. Wertebereich

$$\{-2^{B-1}, \ldots, -1, 0, 1, \ldots, 2^{B-1} - 2, 2^{B-1} - 1\}$$

Wertebereich des Typs `int`

- Repräsentation mit B Bits. Wertebereich

$$\{-2^{B-1}, \ldots, -1, 0, 1, \ldots, 2^{B-1} - 2, 2^{B-1} - 1\}$$

- Auf den meisten Plattformen $B = 32$

Woher kommt gerade diese Aufteilung?

Wertebereich des Typs `int`

- Repräsentation mit B Bits. Wertebereich

$$\{-2^{B-1}, \ldots, -1, 0, 1, \ldots, 2^{B-1} - 2, 2^{B-1} - 1\}$$

- Für den Typ `int` garantiert C++ $B \geq 16$

Woher kommt gerade diese Aufteilung?

Überlauf und Unterlauf

- Arithmetische Operationen (`+`,`-`,`*`) können aus dem Wertebereich herausführen.
- Ergebnisse können inkorrekt sein.

`power8.cpp`: $15^8 = -1732076671$

`power20.cpp`: $3^{20} = -808182895$

- Es gibt *keine Fehlermeldung!*

Der Typ `unsigned int`

- Wertebereich
$$\{0, 1, \ldots, 2^B - 1\}$$
- Alle arithmetischen Operationen gibt es auch für **`unsigned int`**.
- Literale: **`1u, 17u`** ...

Gemischte Ausdrücke

- Operatoren können Operanden verschiedener Typen haben (z.B. `int` und `unsigned int`).

```
17 + 17u
```

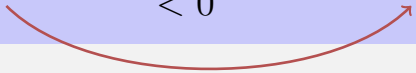
- Solche gemischten Ausdrücke sind vom „allgemeineren” Typ `unsigned int`.
- `int`-Operanden werden *konvertiert* nach `unsigned int`.

Konversion

`int` Wert	Vorzeichen	`unsigned int` Wert
x	≥ 0	x
x	< 0	$x + 2^B$

Konversion

`int` Wert	Vorzeichen	`unsigned int` Wert
x	≥ 0	x
x	< 0	$x + 2^B$



Bei Zweierkomplementdarstellung passiert dabei intern gar nichts

Rechnen mit Binärzahlen (4 Stellen)

Einfache Addition

$$\begin{array}{r} 2 \\ +3 \\ \hline 5 \end{array} \qquad \begin{array}{r} 0010 \\ +0011 \\ \hline 0101 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Einfache Subtraktion

$$\begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array} \qquad \begin{array}{r} 0101 \\ -0011 \\ \hline 0010 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Addition mit Überlauf

$$\begin{array}{r} 7 \\ +9 \\ \hline 16 \end{array} \qquad \begin{array}{r} 0111 \\ +1001 \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Negative Zahlen?

$$\begin{array}{r} 5 \\ +(-5) \\ \hline 0 \end{array} \qquad \begin{array}{r} 0101 \\ ???? \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Einfacher: -1

$$\begin{array}{r} 1 \\ +(-1) \\ \hline 0 \end{array} \qquad \begin{array}{r} 0001 \\ 1111 \\ \hline (1)0000 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Nutzen das aus:

$$\begin{array}{r} 3 \\ +? \\ \hline -1 \end{array} \qquad \begin{array}{r} 0011 \\ +???? \\ \hline 1111 \end{array}$$

Rechnen mit Binärzahlen (4 Stellen)

Invertieren!

$$\begin{array}{r} 3 \\ +(-4) \\ \hline -1 \end{array} \qquad \begin{array}{r} 0011 \\ +1100 \\ \hline 1111 \end{array} \widehat{=} 2^B - 1$$

Rechnen mit Binärzahlen (4 Stellen)

$$\begin{array}{r} a \\ +(-a-1) \\ \hline -1 \end{array} \qquad \begin{array}{r} a \\ \bar{a} \\ \hline 1111 \end{array} \widehat{=} 2^B - 1$$

Rechnen mit Binärzahlen (4 Stellen)

- Negation: Inversion und Addition von 1

$$-a \quad \widehat{=} \quad \bar{a} + 1$$

Rechnen mit Binärzahlen (4 Stellen)

- Wrap-around Semantik (Rechnen modulo 2^B)

$$-a \quad \widehat{=} \quad 2^B - a$$

Warum das funktioniert

Modulo-Arithmetik: Rechnen im Kreis[4]

[4] Die Arithmetik funktioniert auch mit Dezimalzahlen (und auch für die Multiplikation)

Negative Zahlen (3 Stellen)

	a	$-a$
0	000	
1	001	
2	010	
3	011	
4	100	
5	101	
6	110	
7	111	

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001		
2	010		
3	011		
4	100		
5	101		
6	110		
7	111		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	**111**	-1
2	010		
3	011		
4	100		
5	101		
6	110		
7	**111**		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	**110**	-2
3	011		
4	100		
5	101		
6	**110**		
7	111		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	**101**	-3
4	100		
5	**101**		
6	110		
7	111		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	101	-3
4	**100**	**100**	-4
5	101		
6	110		
7	111		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	101	-3
4	100	100	-4
5	101		
6	110		
7	111		

Negative Zahlen (3 Stellen)

	a	$-a$	
0	000	000	0
1	001	111	-1
2	010	110	-2
3	011	101	-3
4	100	100	-4
5	101		
6	110		
7	111		

Das höchste Bit entscheidet über das Vorzeichen.