2. 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

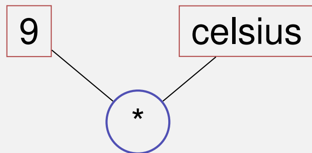
Klammerung ergibt Ausdrucksbaum

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

Ausdrucksbäume

Klammerung ergibt Ausdrucksbaum

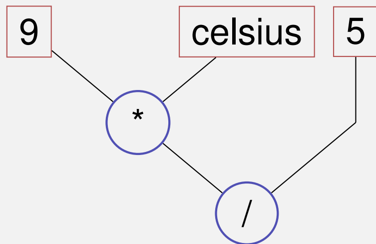
```
(9 * celsius) / 5 + 32
```



Ausdrucksbäume

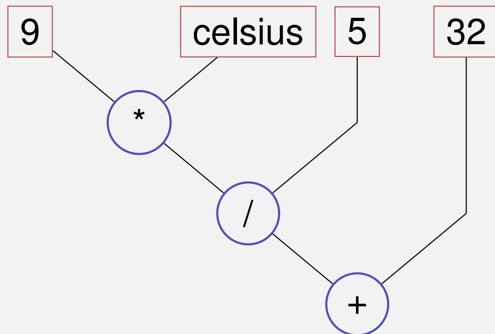
Klammerung ergibt Ausdrucksbaum

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



Ausdrucksbäume

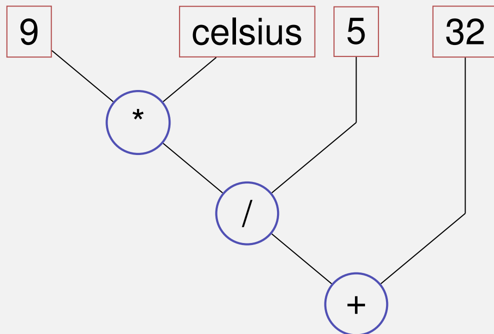
Klammerung ergibt Ausdrucksbaum



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

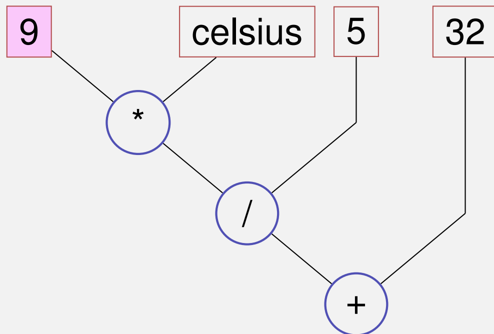
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

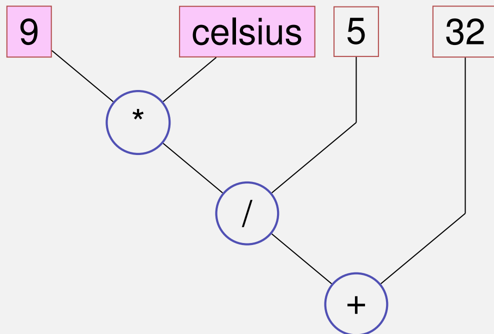
`9 * celsius / 5 + 32`



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

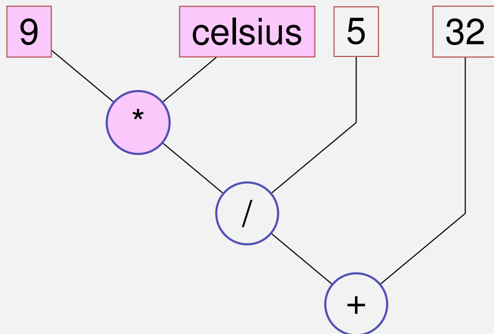
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

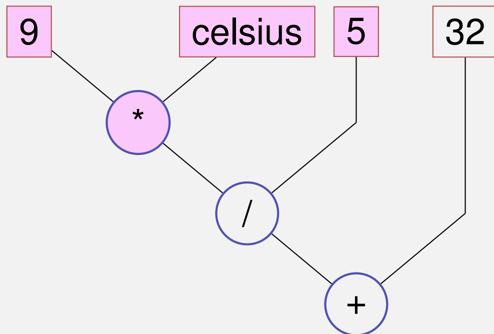
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

„Von oben nach unten" im Ausdrucksbaum

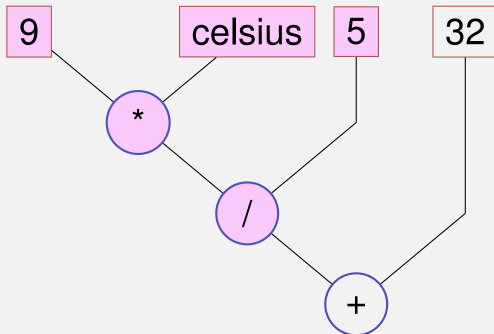
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

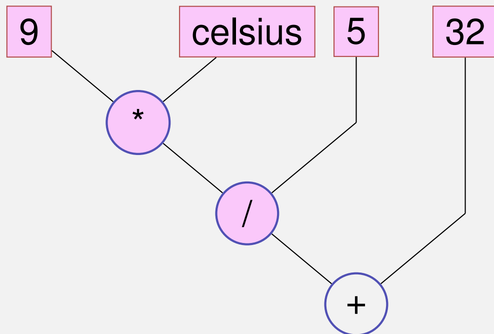
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

„Von oben nach unten“ im Ausdrucksbaum

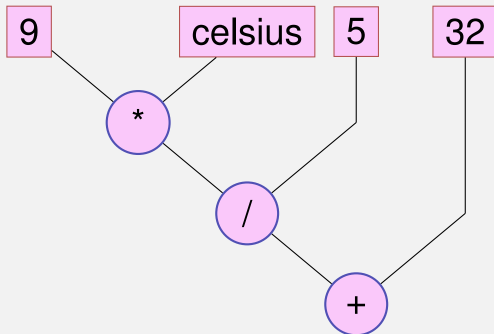
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

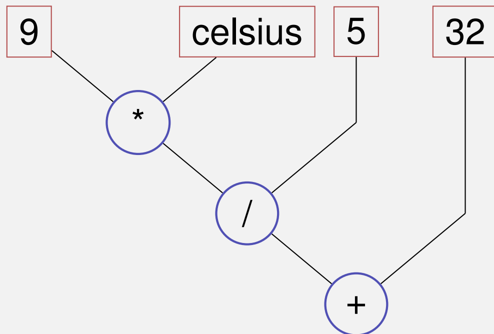
„Von oben nach unten“ im Ausdrucksbaum

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



Auswertungsreihenfolge

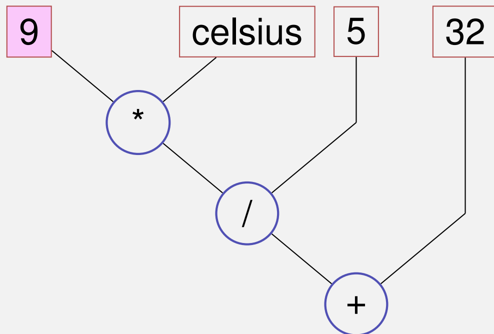
Reihenfolge nicht eindeutig bestimmt:



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

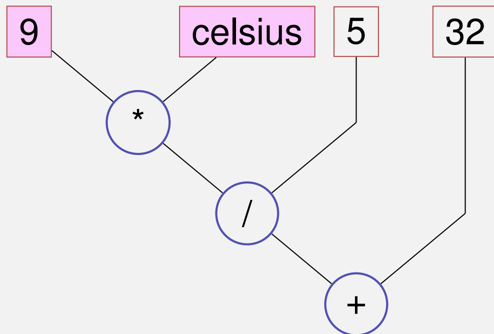
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

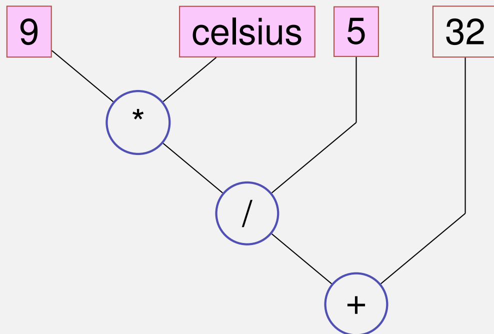
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

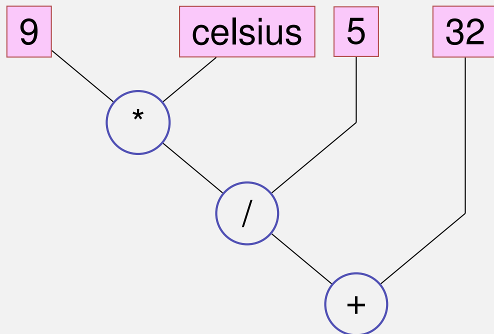
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

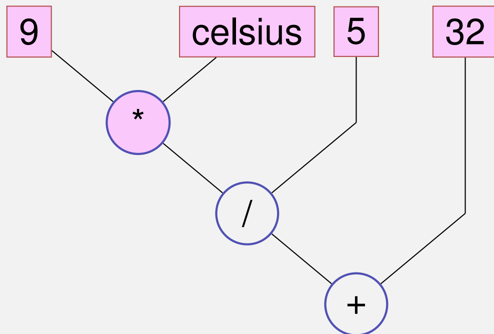
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

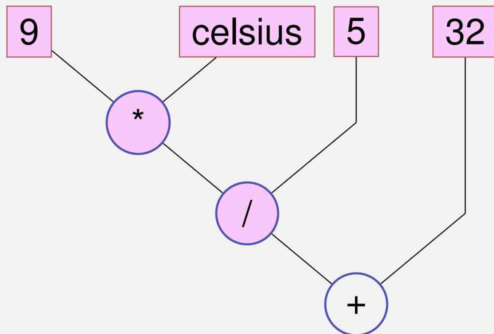
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

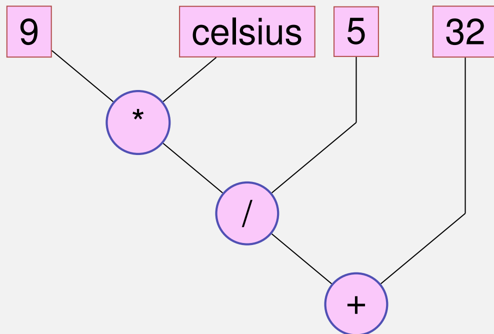
```
9 * celsius / 5 + 32
```



Auswertungsreihenfolge

Reihenfolge nicht eindeutig bestimmt:

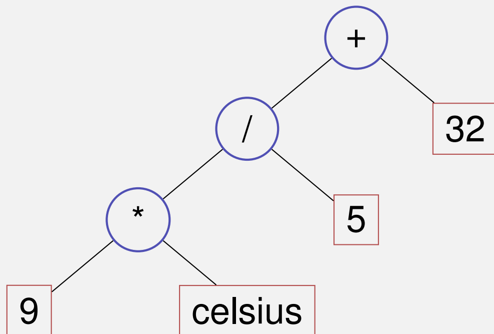
```
9 * celsius / 5 + 32
```



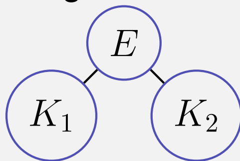
Ausdrucksbäume – Notation

Üblichere Notation: Wurzel oben

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

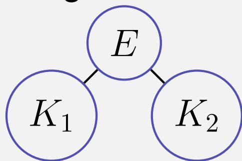


Auswertungsreihenfolge – formaler

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

Auswertungsreihenfolge – formaler

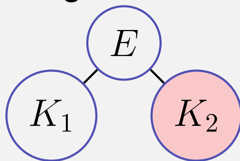
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

Auswertungsreihenfolge – formaler

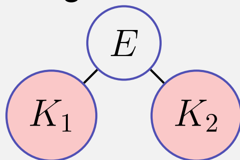
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

Auswertungsreihenfolge – formaler

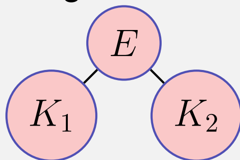
- Gültige Reihenfolge: Jeder Knoten wird erst *nach* seinen Kindern ausgewertet.



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

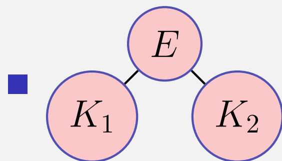
Auswertungsreihenfolge – formaler

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



C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

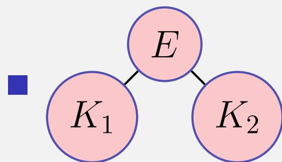
Auswertungsreihenfolge – formaler

- 

C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

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

Auswertungsreihenfolge – formaler

- 

C++: anzuwendende gültige Reihenfolge nicht spezifiziert.

- Beispiel eines „schlechten Ausdrucks“: `a*(a=2)`

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
Modulo	%	2	14	links
Addition	+	2	13	links
Subtraktion	−	2	13	links

Einschub: Zuweisungsausdruck – nun genauer

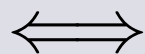
- Bereits bekannt: `a = b` bedeutet
  Zuweisung von `b` (R-Wert) an `a` (L-Wert).
  Rückgabe: L-Wert

Einschub: 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` ?

Einschub: 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)`

Einschub: Zuweisungsausdruck – nun genauer

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

Beispiel Mehrfachzuweisung:

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

Division

- Operator / realisiert ganzzahlige Division

```
5 / 2 hat Wert 2
```

Division

- Operator / realisiert ganzzahlige Division

`5 / 2` hat Wert 2

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32

15 degrees Celsius are 59 degrees Fahrenheit
```

Division

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32

15 degrees Celsius are 59 degrees Fahrenheit
```

Division

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32

15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent...

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

Division

- In `fahrenheit.cpp`

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

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

- Mathematisch äquivalent…

```
1 * celsius + 32
```

Division

- In `fahrenheit.cpp`

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

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

- Mathematisch äquivalent...

```
15 + 32
```

Division

- In `fahrenheit.cpp`

```
9 * celsius / 5 + 32

15 degrees Celsius are 59 degrees Fahrenheit
```

- Mathematisch äquivalent…

```
47
```

Division

- 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
```

Präzisionsverlust

Richtlinie

- Auf möglichen Präzisionsverlust achten
- Potentiell verlustbehaftete Operationen möglichst spät durchführen um „Fehlereskalation“ zu vermeiden

Division und Modulo

- Modulo-Operator berechnet Rest der ganzzahligen Division

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

Division und Modulo

- Modulo-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

- relativ lang

Inkrement und Dekrement

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

Nachteile

- relativ lang
- `expr` wird zweimal ausgewertet
  - Später: L-wertige Ausdrücke deren Auswertung „teuer“ ist

Inkrement und Dekrement

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

Nachteile

- relativ lang
- `expr` wird zweimal ausgewertet
  - Später: L-wertige Ausdrücke deren Auswertung „teuer“ ist
  - `expr` könnte einen Effekt haben (aber sollte nicht, siehe Richtlinie)

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 $b_n \cdot 2^n + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0$

Binäre Zahlendarstellungen

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

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

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

Binäre Zahlendarstellungen

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

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

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

Beispiel: `101011`

Binäre Zahlendarstellungen

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

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

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

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 $b_n \cdot 2^n + \cdots + b_1 \cdot 2 + b_0$

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.

Niedrigstes Bit, Least Significant Bit (LSB)

Höchstes Bit, Most Significant Bit (MSB)

Rechentricks

- Abschätzung der Grössenordnung von Zweierpotenzen[2]:

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

[2]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[2]:

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

[2]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[2]:

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

[2] 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

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

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?

„Für Programmierer und Techniker“

(Bedienungsanleitung Schachcomputer *Mephisto II*, 1981)

Beispiele:

8200

7F00

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

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:

$$ABCD = (A\text{x}16^3) + (B\text{x}16^2) + (C\text{x}16^1) + (D\text{x}16^0)$$

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:
$(14\text{x}16^0) + (5\text{x}16^1) + (0\text{x}16^2) + (0\text{x}16^3) = 14+80+0+0 =$
$= +94$ *Punkte.*

7F80

d) Anzeige ***7F80***
(7=-1; F=15) Umrechnung wie folgt:
$(0\text{x}16^0) + (8\text{x}16^1) + (15\text{x}16^2) - (1\text{x}16^3) = 0+128+3840-4096 =$

Beispiel: Hex-Farben

#00FF00

r g b

Beispiel: Hex-Farben

Beispiel: Hex-Farben

#808080

r g b

Beispiel: Hex-Farben

#FF0050

r g b

Wertebereich des Typs `int`

```
// 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`

```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;
}
```

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

Wertebereich des Typs `int`

```
// 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;
}
```

```
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

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

Woher kommt gerade diese Aufteilung?

- Auf den meisten Plattformen $B = 32$

Wertebereich des Typs `int`

- Repräsentation mit B Bits. Wertebereich

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

  Woher kommt gerade diese Aufteilung?

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

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 (folgt) 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

Dezimal	Binär
5	0101
-3	-0011
2	0010

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)

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

Rechnen mit Binärzahlen (4 Stellen)

- Negation: Inversion und Addition von 1

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

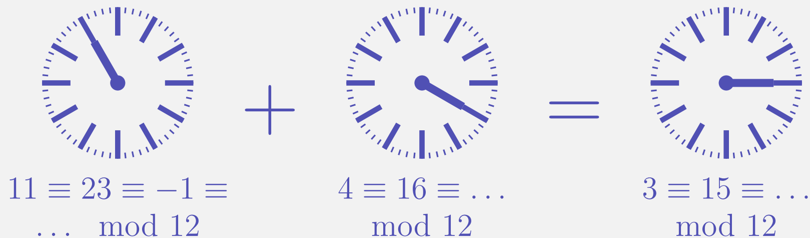
Rechnen mit Binärzahlen (4 Stellen)

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

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

Warum das funktioniert

Modulo-Arithmetik: Rechnen im Kreis[3]



[3] 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 *und* es trägt zum Zahlwert bei.