TUHH
Technische Universität Hamburg-Harburg

Institute of Embedded Systems

Hello World!

Eine Einführung in das Programmieren

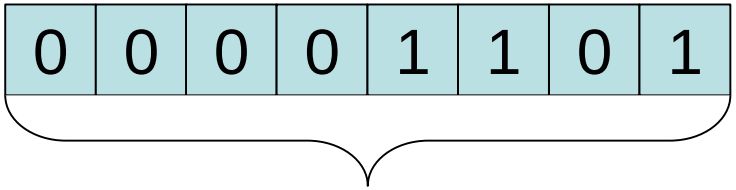
Variablen

Görschwin Fey

Institute of Embedded Systems
Hamburg University of Technology

Wie werden Daten in Programmen gespeichert und manipuliert?

Darstellung von Werten



8 Register bzw. 8 Bit
= 1 Byte

=

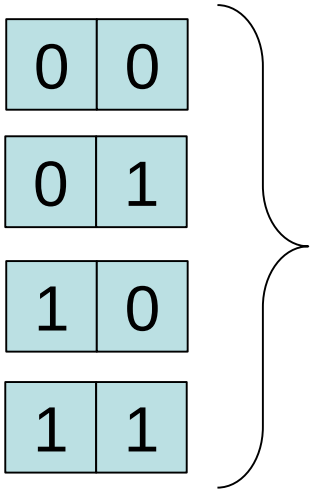
„8 mal 2 Möglichkeiten“
zur Darstellung von
Werten

$$2^8 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 256$$

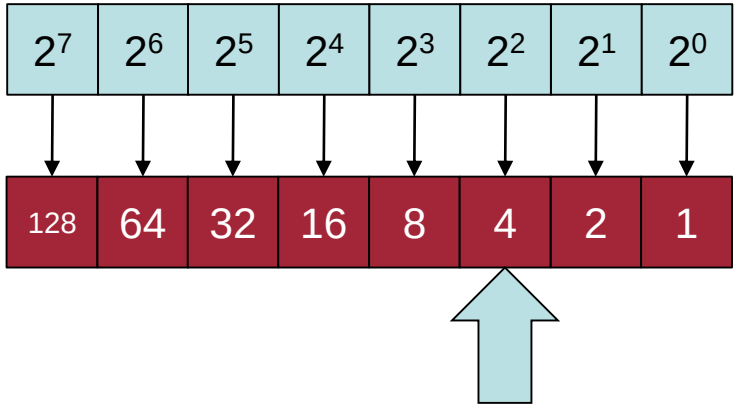
Erst durch einen Datentyp wird der Binärdarstellung ein Wert zugeordnet!

Natürliche Zahlen

Jedem Bit wird als Wertigkeit die Anzahl der vorherigen möglichen Belegungen zugeordnet.



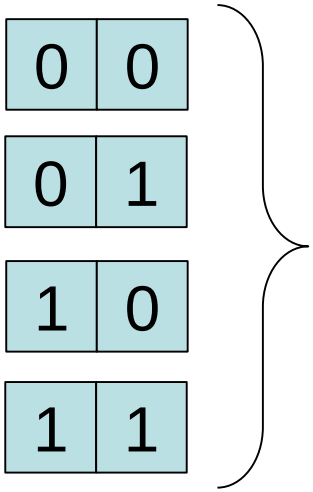
„4 Mögliche“ Belegungen bei 2 Bits



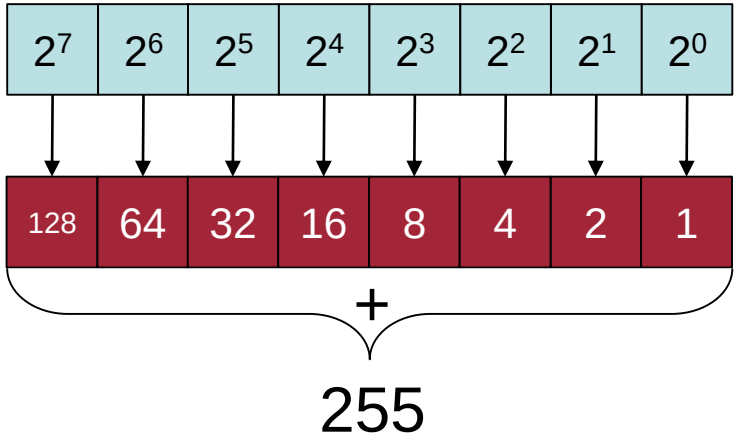
Das 3. Bit hat Wertigkeit 4

Natürliche Zahlen

Jedem Bit wird als Wertigkeit die Anzahl der vorherigen möglichen Belegungen zugeordnet.

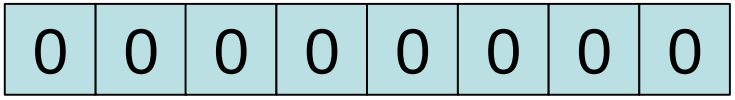


„4 Mögliche“ Belegungen bei 2 Bits



$$2^8 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 256$$

Auch die „0“ ist ein Wert:

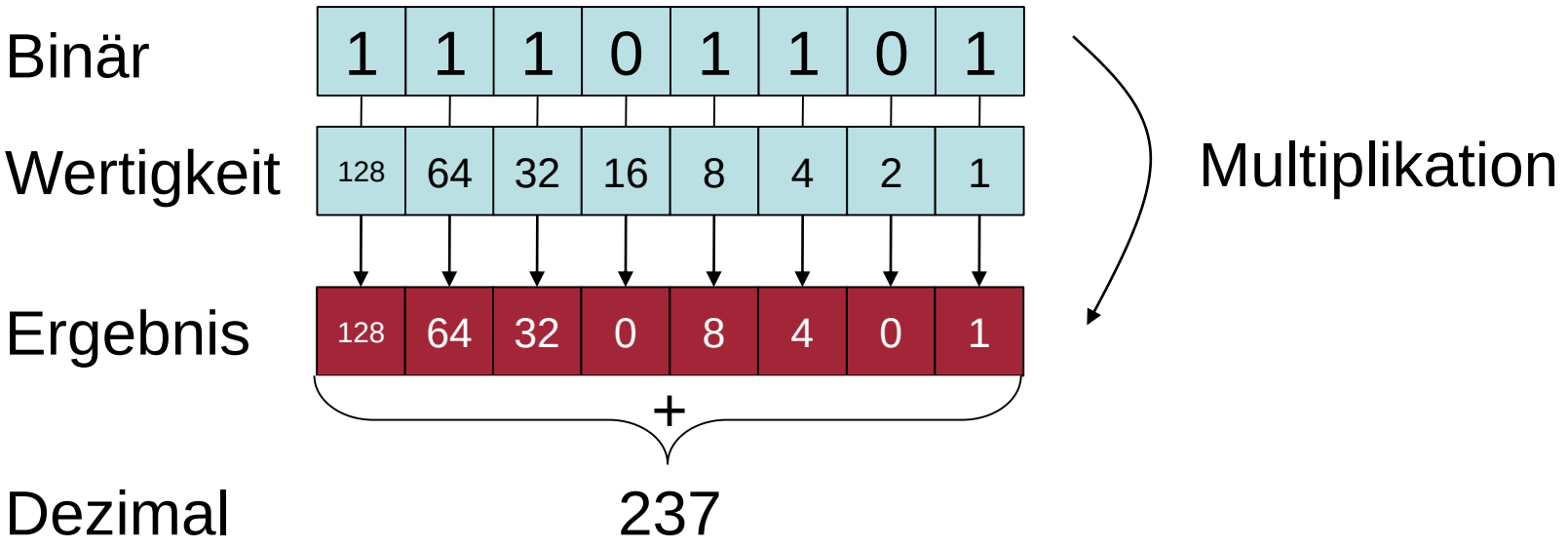


Natürliche Zahlen

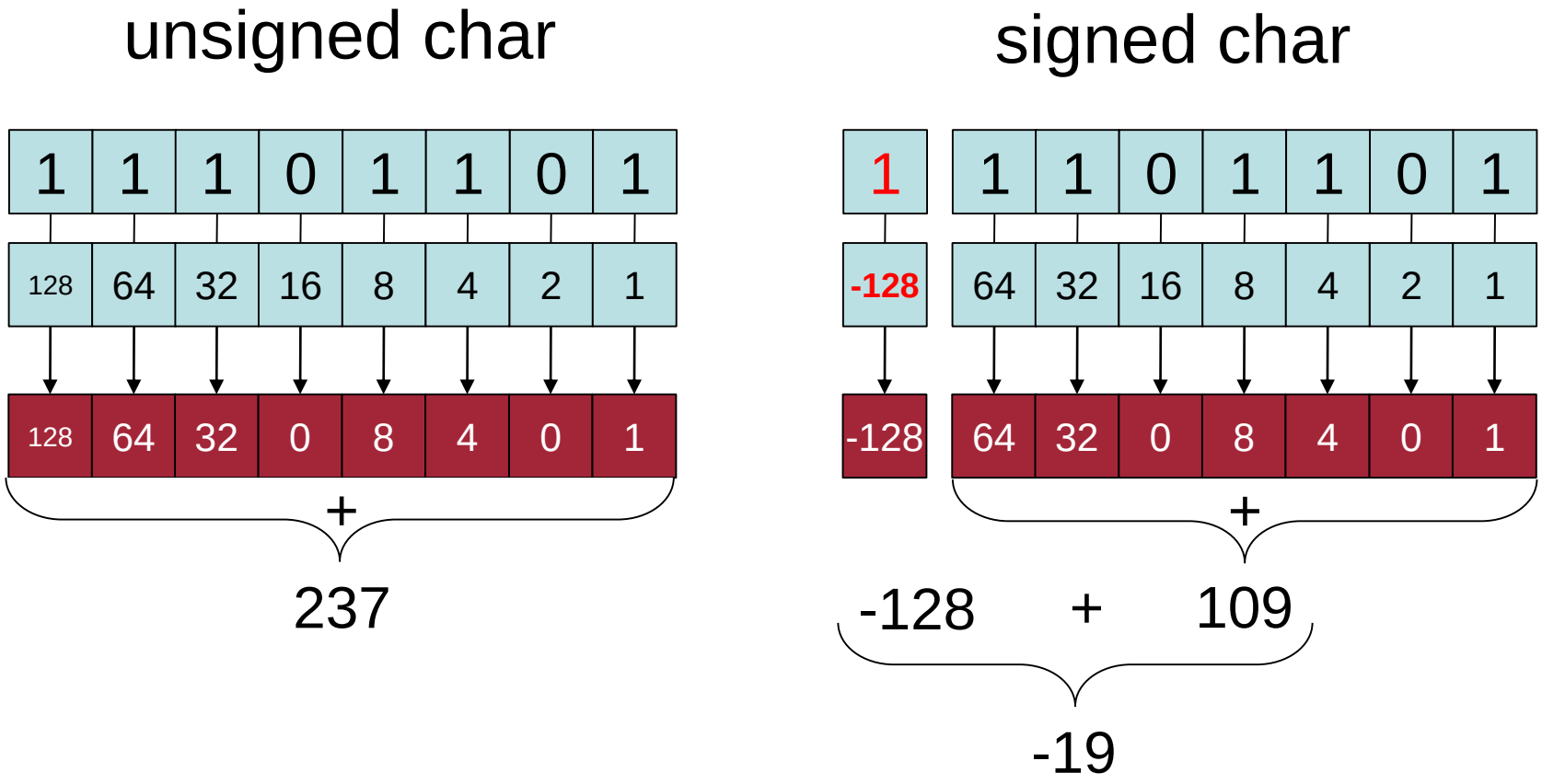
Zur Umrechnung der Binärdarstellung in das Dezimalzahlsystem werden…

1. die Bits mit ihrer Wertigkeit multipliziert
2. die Ergebnisse miteinander addiert

Mit 8 Bit lassen sich positive Werte nur im Bereich 0 bis 255 darstellen.



Ganze Zahlen



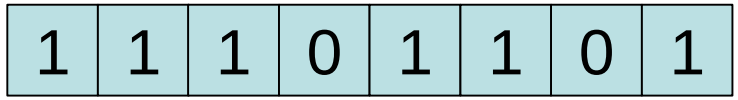
Erinnerung: 2er-Komplement $v_2(z_B) = (-2)^n * z_n + \sum_{j=0}^{n-1} z_j B^j$

Tatsächliche Darstellung ist vom Rechner abhängig

Datengröße

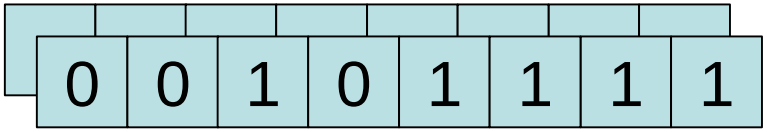
Der Arbeitsspeicher ist in Bytes organisiert.
Wenn ein Byte nicht reicht, dann nimmt man einfach mehr!

8 Bits
1 Byte



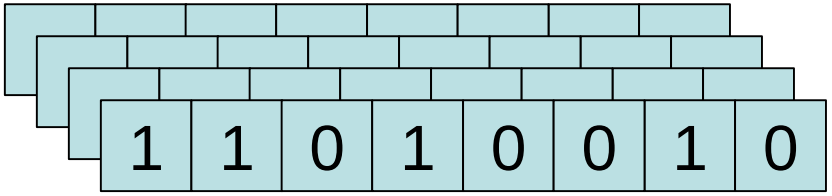
256 Werte

16 Bits
2 Byte



65536 Werte

32 Bits
4 Byte



4294967296 Werte

Primitive Datentypen in C++

`sizeof(Typ)` liefert Größe von `Typ` in Byte

Datentyp	Bit	Wertebereich Anfang	Wertebereich Ende
char	8	-128	127
unsigned char	8	0	255
short	16	-32 768	32 767
unsigned short	16	0	65 532
int	32	-2 147 483 648	2 147 483 647
unsigned int	32	0	4 294 967 295
long	64	-9 233 372 036 854 755 808	9 233 372 036 854 755 807
unsigned long	64	0	18 466 744 073 701 511 613
float	32	$1.2 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$
double	64	$2.2 \cdot 10^{-308}$	$1.8 \cdot 10^{308}$

ASCII-Standard

Um Schriftzeichen bzw. Symbole abzuspeichern wird eine Zuordnung bzw. festgelegte Codierung benötigt.

Weit verbreitet ist der ASCII-Standard → Ein Wert wird eindeutig einem Zeichen zugeordnet.

Beispiel:	7A	entspricht	„z“
	80	entspricht	„€“

Auch Sonderzeichen wie z.B. „Zeilenumbruch“ haben einen Code

Im Internet lassen sich diverse ASCII-Tabelle finden, in denen die Zuordnung beschrieben wird.

Variablen

Das Anlegen der Variable erfolgt in zwei Schritten:

1. **Deklaration** – Speicherplatz für die Variable reservieren

```
Datentyp Bezeichner;
```

- Datentyp legt fest, wieviel Speicher reserviert werden muss
- Die Variable braucht einen Namen bzw. einen Bezeichner

2. **Initialisieren** – Der Variablen den ersten Wert zuweisen

```
Bezeichner = Wert;
```

- Vor der Initialisierung hat die Variable keinen wohldefinierten Wert
- Der Inhalt des reservierten Speichers wird nach der Vorschrift des Datentyps interpretiert

Variablen

Die Variable kann auch in einem Schritt deklariert und initialisiert werden.

```
Datentyp Bezeichner = Wert;
```

Beispiel: getrennte Deklaration und Initialisierung

```
int a;
a = 3;
```

Beispiel: gemeinsame Deklaration und Initialisierung

```
int a = 3;
```

Beide Varianten funktionieren auch mit mehreren Variablen gleichzeitig:

```
int a,b;
a = 3;
b = 3;
```

```
int a = 3, b = 3;
```

Bezeichner

1. Der Bezeichner kann aus den folgenden Zeichen gebildet werden.

```
$ a b c d e f g h i j k l m n o p q r s t u v w x y z
_ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

2. Der Bezeichner **KANN** Ziffern enthalten, darf aber **NICHT** mit einer Ziffer beginnen.

```
1 2 3 4 5 6 7 8 9 0
```

3. Schlüsselwörter von C++ sind **KEINE** zulässigen Bezeichner.

Beispiel: Bezeichner

Die Umlaute ä, ö, ü und ß sind in Bezeichnern nicht erlaubt.

Zulässige Bezeichner

```
_unterstrich
__2Unterstriche
whileTrue
X
kuehlung_an
…
```

Nichtzulässige Bezeichner

```
sonderMüll
kunde_straße
Ölstand
3Rad
while
…
```

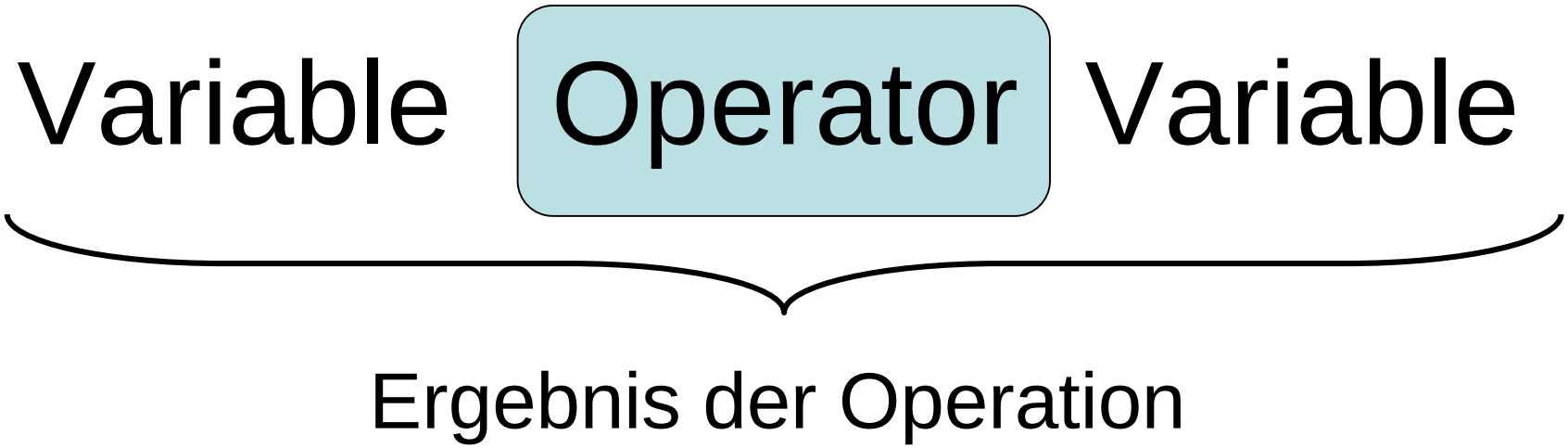
Tipp: Programme sind besser lesbar und verständlicher durch

- aussagekräftige Bezeichner wie z.B. `reifenDruck` oder `speed_kmh`
- englischsprachige Bezeichner bevorzugen; so werden Umlaute vermieden

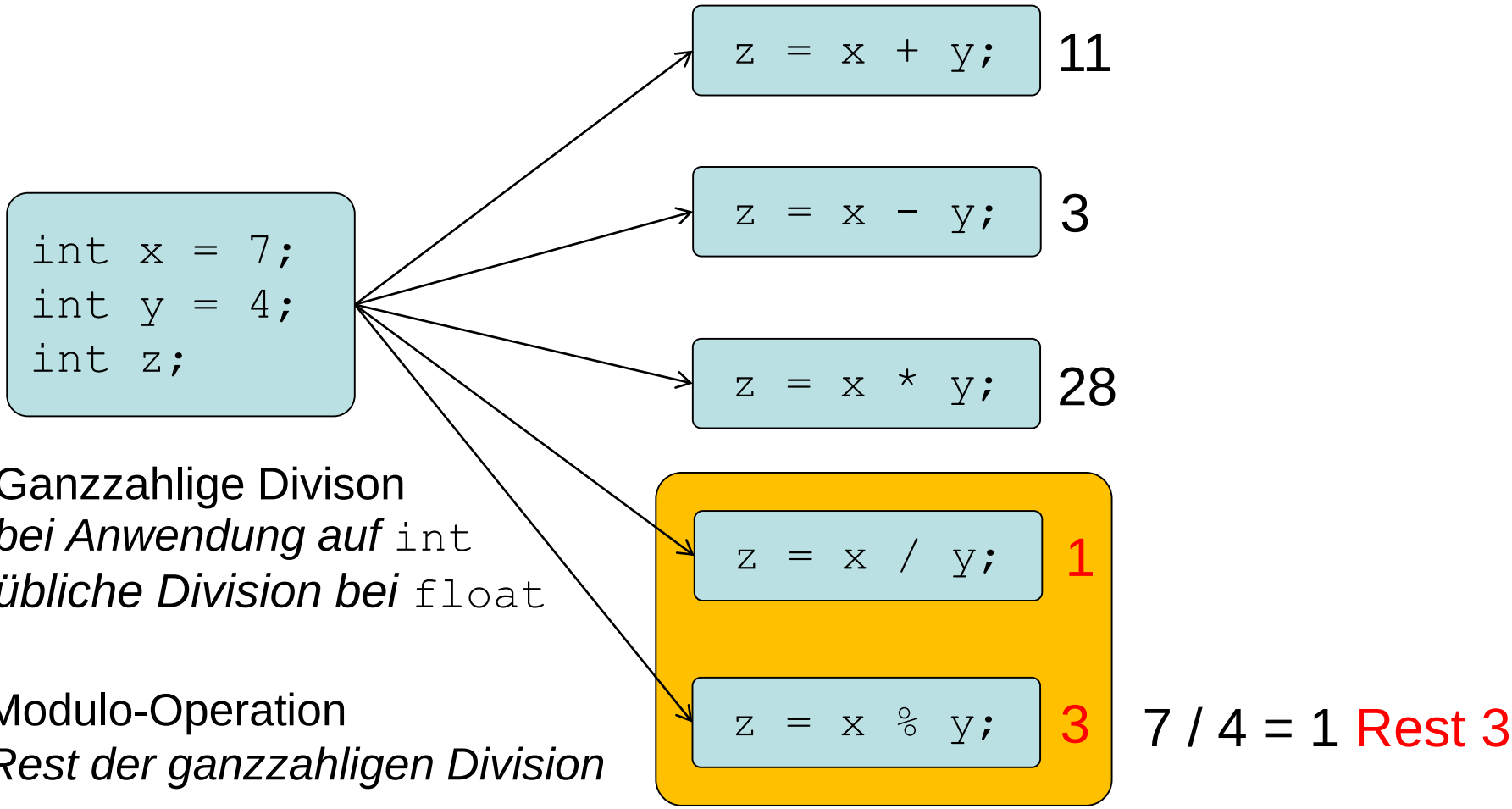
Was ist ein Operator?

- Dient der Manipulation von Variablenwerten und liefert ein Ergebnis
- Funktion, die von einem Operator implementiert wird, ist abhängig von Typen der Variablen, auf die er angewendet wird

- Beispiel:
  Binärer Operator wird auf zwei Variablen angewendet:



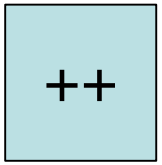
Rudimentäre Operationen



Ganzzahlige Divison
bei Anwendung auf `int`
übliche Division bei `float`

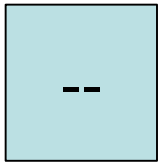
Modulo-Operation
Rest der ganzzahligen Division

Increment und Decrement



Inkrement

Addiert 1 zu einer Variablen



Dekrement

Subtrahiert 1 von einer Variablen

Beispiel:

```
int a = 5;
a++; // a ist 6
a--; // a ist 5
```

Das Addieren / Subtrahieren erfolgt nach der Anweisung.

```
int a = 5, b = 5, c;
c = 3 + --a; // c ist 7; a ist 4
c = 3 + b--; // c ist 8; b ist 4
```

Subtraktion vor der Anweisung!

Vergleiche und Relationen

==

Gleichheit

!=

Ungleichheit

```
int x = 7;
int y = 4;
x == y;
x != y;
```

Inhalt von x und y vergleichen.

<

echt kleiner

>

echt größer

<=

kleiner gleich

>=

größer gleich

```
3 > 3; //False
3 >= 3; //True
```

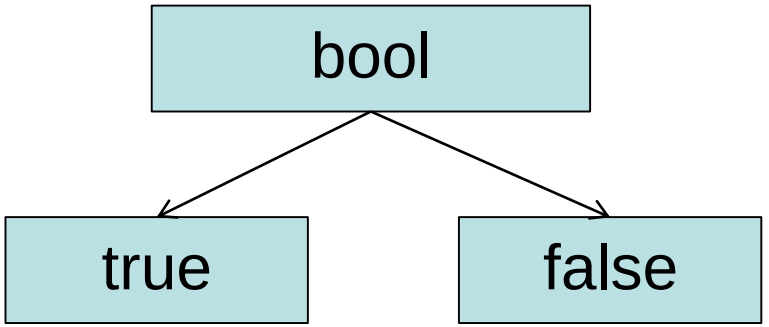
Reihenfolge der Zeichen wie gesprochen
z. B. kleiner (<) gleich (=)

Was ist das Ergebnis einer Vergleichsoperation?

Logisch Vergleichen

Ein Datentyp fehlt noch…

Wahrheitswerte werden in Variablen vom Typ `bool` (kurz für boolean) abgespeichert



Variablen vom Typ `bool` können logisch miteinander verglichen werden

&&	||	!
Logisches AND	Logisches OR	Logisches NOT

Vieles geht! Aber was kommt heraus?

```
float a= 2.5;
float b= 3.5;
float c= b/a;
int d= b/a;

cout <<"c " <<c <<endl;
cout <<"d " <<d <<endl;

int e= c;
cout <<"e " <<e <<endl;
```

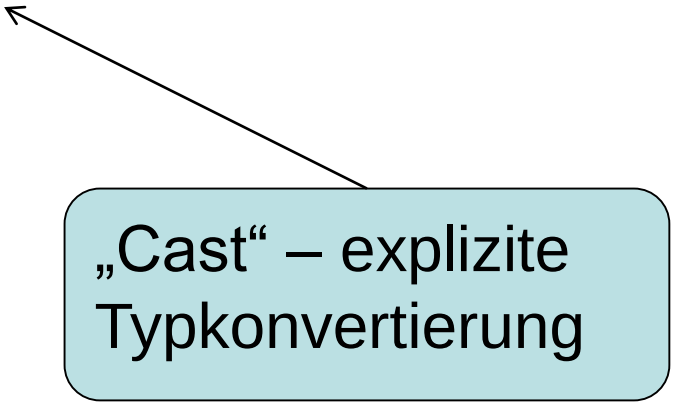
Ausgabe:

```
c 1.4
d 1
e 1
```

Vorsicht bei Typkonvertierungen

```
unsigned char o= 255;
signed char p= (signed char)o;

int q= o;
cout <<"q " <<q <<endl;
q= p;
cout <<"q " <<q <<endl;
```



Ausgabe:

```
q 255
q -1
```

Welcher Operator hat Vorrang?

Die Operatoren haben eine Priorität mit der sie ausgeführt werden. Hier mit absteigender Priorität:

```
(…) ++ -- (Vorzeichen +/-) ! ~ * / % + - << >> < <= > >= == != & ^ | && || = += -= *= /= %= >>= <<= &= |= ^=
```

Durch das Setzen von Klammern (…), kann die Interpretation eines Ausdrucks beeinflusst werden. Klammern haben immer die höchste Priorität.

Wenn eine Unsicherheit bleibt, wie ein Ausdruck interpretiert wird, sollten immer Klammern gesetzt werden!

Zusammenfassung Operationen und Operatoren

- Seien

```
int a, b=1, c=2;
bool r= false, l;
```

- Arithmetisch **+, -, *, /**

```
a= 5;    // Zuweisung
a= b+c; // Addition
a+= b;  // entspricht a= a+b;
```

- Vergleiche **==, <=, >=**

```
r= a >= b; // -> r= true
```

- Logisch **&&, ||, !**

```
l= !r; // r==true -> l= false
```

Kommentare

Mit zwei Schrägstrichen (//) wird ein einzeiliger Kommentar eingeleitet.

```
// Kommentar
```

Mit (/*) beginnt ein mehrzeiliger Kommentar, der mit (*/) wieder beendet wird.

```
/*
     Kommentare
*/
```

Sequenzen

Mehrere Anweisungen werden mit geschweiften Klammern zu einem Block resp. zu einer Sequenz von Anweisungen zusammengefasst.

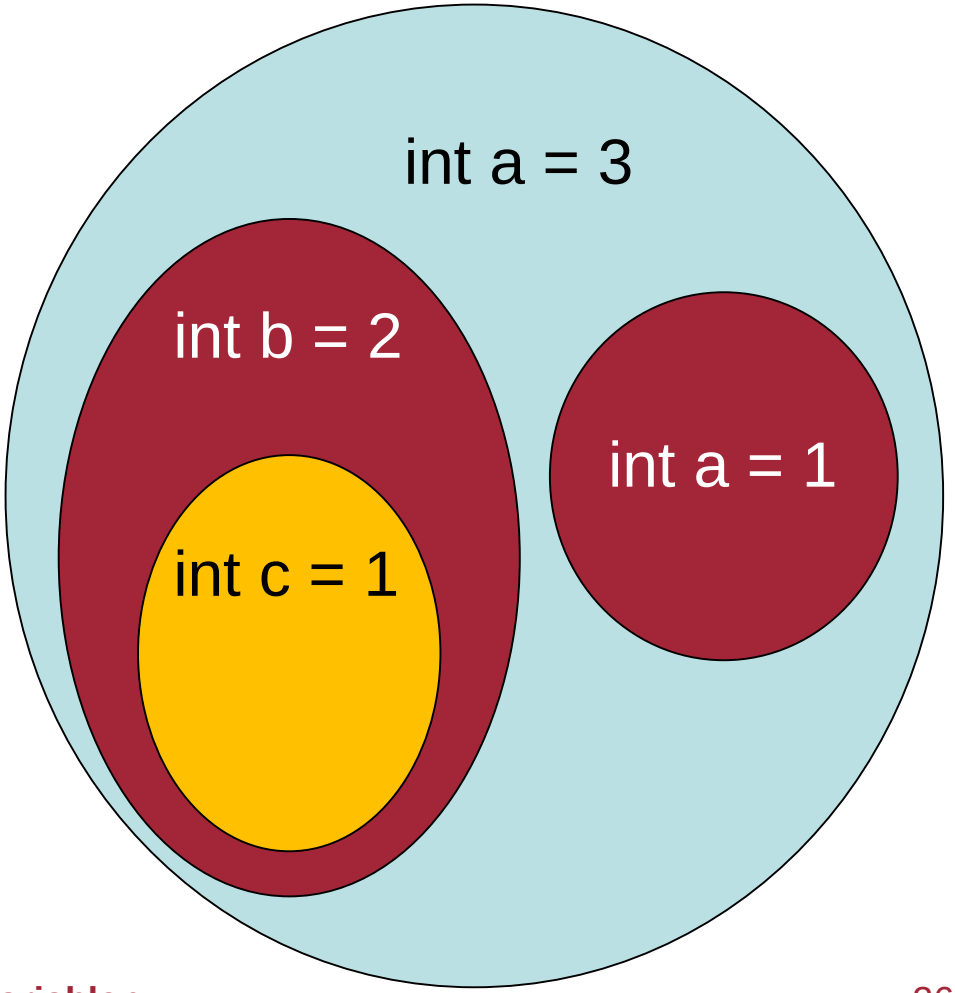
```
{
        Anweisung 1;
        Anweisung 2;
        …
        Anweisung n;
}
```

Zur besseren Lesbarkeit eines Programms, werden alle Anweisungen innerhalb des Blocks mit einem „Tab“ eingerückt.

Scopes

Jede Sequenz von Anweisungen hat einen eigenen Geltungsbereich für Variablen

```
int a = 3;
{
    int b = 2;
    {
        int c = 1;
    }
}
{
    int a = 1;
}
```



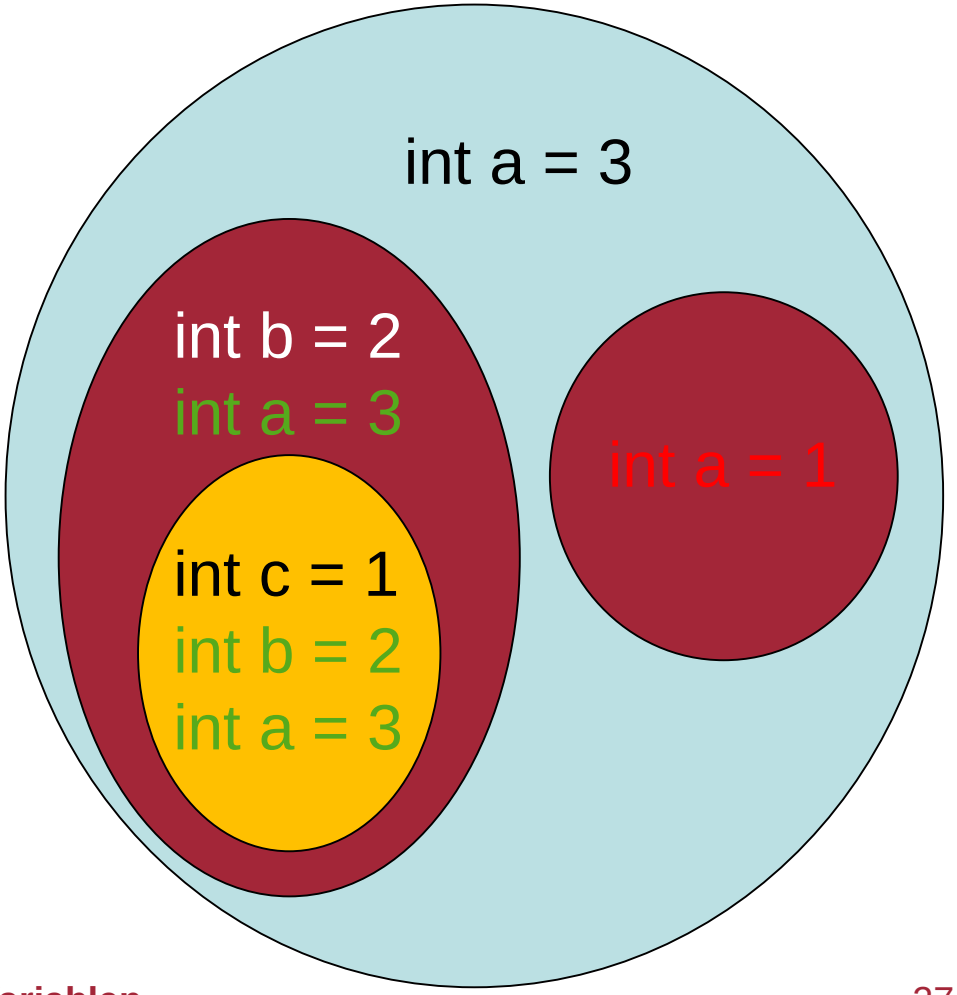
Scopes

Jede Sequenz von Anweisungen hat einen eigenen Geltungsbereich für Variablen

Die Variable ist in dem Scope der Sequenz, in der sie definiert wurde, und in den eingebetteten Sequenzen sichtbar.

Erneute Definition einer Variable überschreibt die Sichtbarkeit der äußeren Variable.

Erneute Definition innerhalb eines Scopes ist nicht erlaubt.



Beispiel: Scopes

```
{
    int b = 2;
}
int a = b;
```

➔ Fehler: Variable b ist nicht definiert!

```
int a = 4;
{
    a = 3;
}
cout << a;
```

Es wird 3 ausgegeben!

```
int a = 4;
{
    int a = 3;
}
cout << a;
```

Es wird 4 ausgegeben!
Schlecht lesbar -> vermeiden

Fortsetzung: Scopes

```
int a = 4;
{
    int a = 3;
}
cout << a;
```

Noch einmal dasselbe Beispiel:

- Nach Verlassen der Sequenz kann auf die Variable a mit dem Wert 3 nicht mehr zugegriffen werden.
- Die Variable a mit dem Wert 3 wird nach der Sequenz aus dem Speicher gelöscht.

Zusammenfassung

- Vorsicht bei Typkonvertierungen
- Hardware-nahe und speichereffiziente Programmierung durch Pointer möglich
- Vorsicht bei direkten Speicherzugriffen

Übung

1. Füge eine Variablen-Deklaration, Wertezuweisung und Ausgabe im HelloWorld-Programm ein.
2. Definiere ein Rechteck mit Breite `breite` und Länge `laenge`, berechne den Flächeninhalt und Umfang, gebe beides aus. Nutze einmal Variablen vom Typ `int` und einmal vom Typ `float`.
3. Und nochmal für einen Kreis!
4. Was passiert bei unzulässigen Variablen-Namen?
5. Was passiert bei der Zuweisungen zwischen unterschiedlichen Datentypen?

   `int` → `float; float` → `int;`

   `char` → `float, int` → `bool`
6. Was passiert bei arithmetischen Operationen, an denen unterschiedliche Datentypen beteiligt sind?