# Kommentare in C

```
int a[8] = {0}, b[4], c[4];
int s = 8, t = 4;

while(s--){ a[s] = s*(s+1)

while(t--){

        b[t] = a[2*t];
        c[t] = a[2*t+1];
}
```

# Kommentare in C

**Keine Ahnung was ich vor einem Jahr gemacht habe.**
**Ich verstehe es nicht – ich schreibe es nochmal …..**

**Vielleicht hilft ja ein gut kommentierter Quellcode …..**
**….. nur ein Tipp fürs nächste Mal …..**

# Kommentare in C

## Was ist ein Kommentar?

**Mitteilungen im Quellcode, die vom Compiler ignoriert werden.**

**Hinweise, damit der Quellcode verständlicher wird.**

## C kennt zwei Möglichkeiten Kommentare in den Quellcode einzufügen …..

```
/*
while(t--){

        b[t] = a[2*t];          // int i=8;
        c[t] = a[2*t+1];
}
*/
```

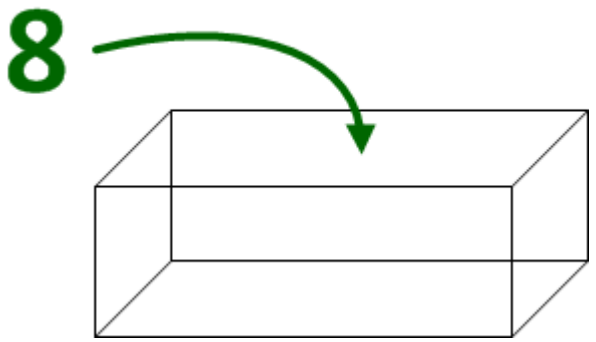
# Kommentare in C

```c
int main()
{
  printf("Variante 1: Blockweises Auskommentieren");

  /*

  Der Standard C-Kommentar beginnt mit dem
  kombinierten Stern-Schrägstrich Symbol /*
  und endet mit dem Schrägstrich-Stern Symbol
  (siehe Kommentarende)

  Alle Texte zwischen diesen beiden Symbolen
  werden vom Compiler ignoriert.

  */

  system("PAUSE");
  return 0;
}
```

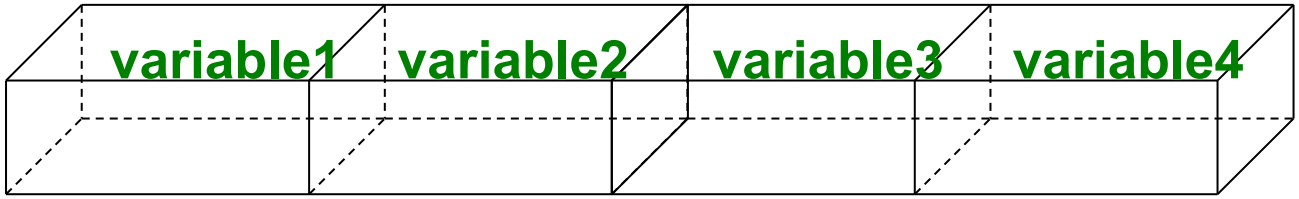
# Kommentare in C

```c
int main()
{
  // Dieser Kommentar beginnt mit zwei Schrägstrichen

  printf("Variante 2: Zeilenweisen Auskommentieren");

  // und kommentiert immer nur bis zum Zeilenende aus
  // Daher beachtet vom Compilerden printf()-Befehl

  printf("Das Programm ist jetzt zu Ende.\n");

  // Da war ja noch eine Ausgabe

  system("PAUSE");
  return 0;
}
```

# Variablen in C




# *Der Einsatz von Variablen ……*




## *Aus welchen Gründen sollte ich eine Variable verwenden?*

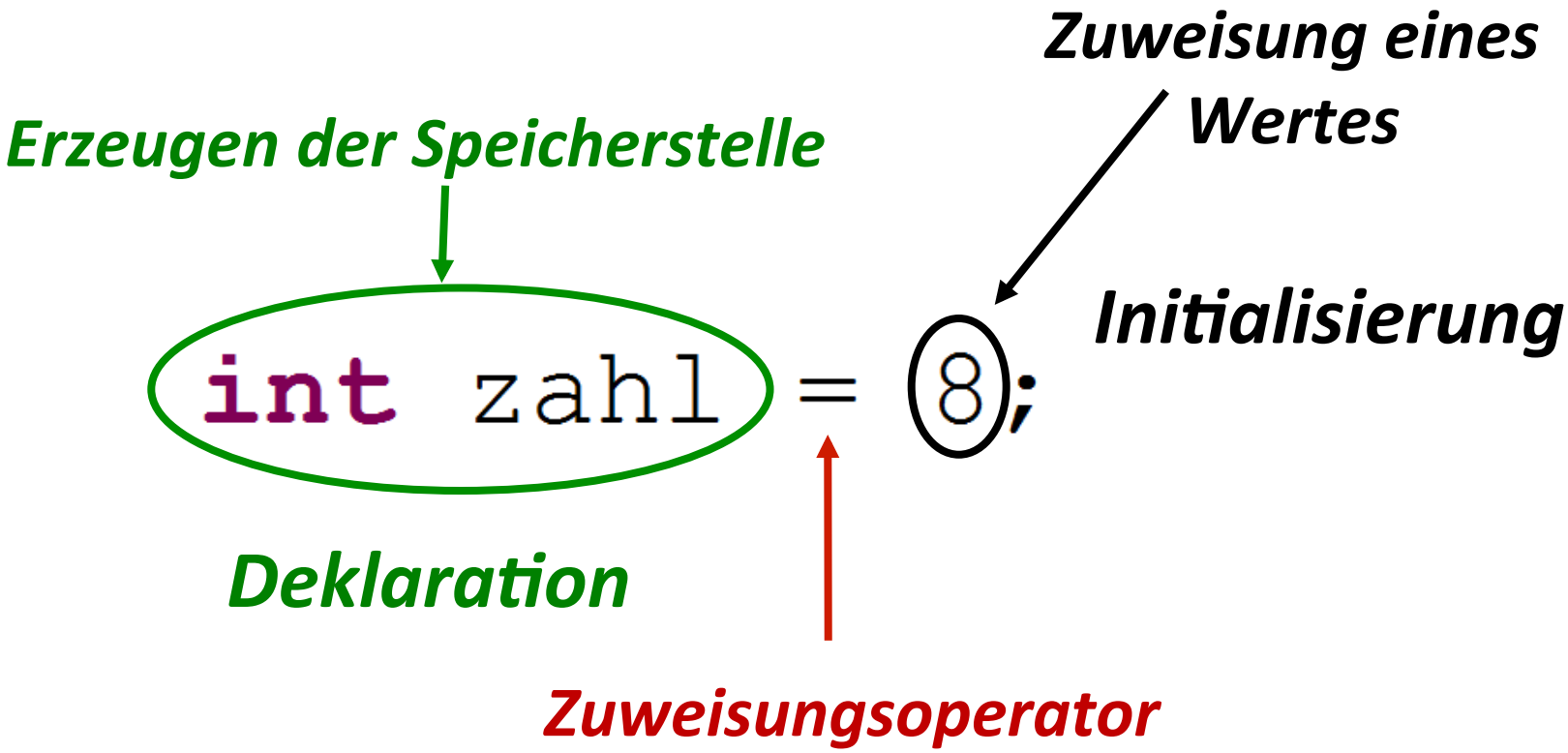
***In einer Variable können Werte, Zeichenketten oder Adressen abgelegt werden, die jederzeit abrufbar sind***

## *Wie kann man den Begriff einer Variablen beschreiben?*

***Symbol, das einer Speicherstelle im Arbeitsspeicher des Rechners entspricht***

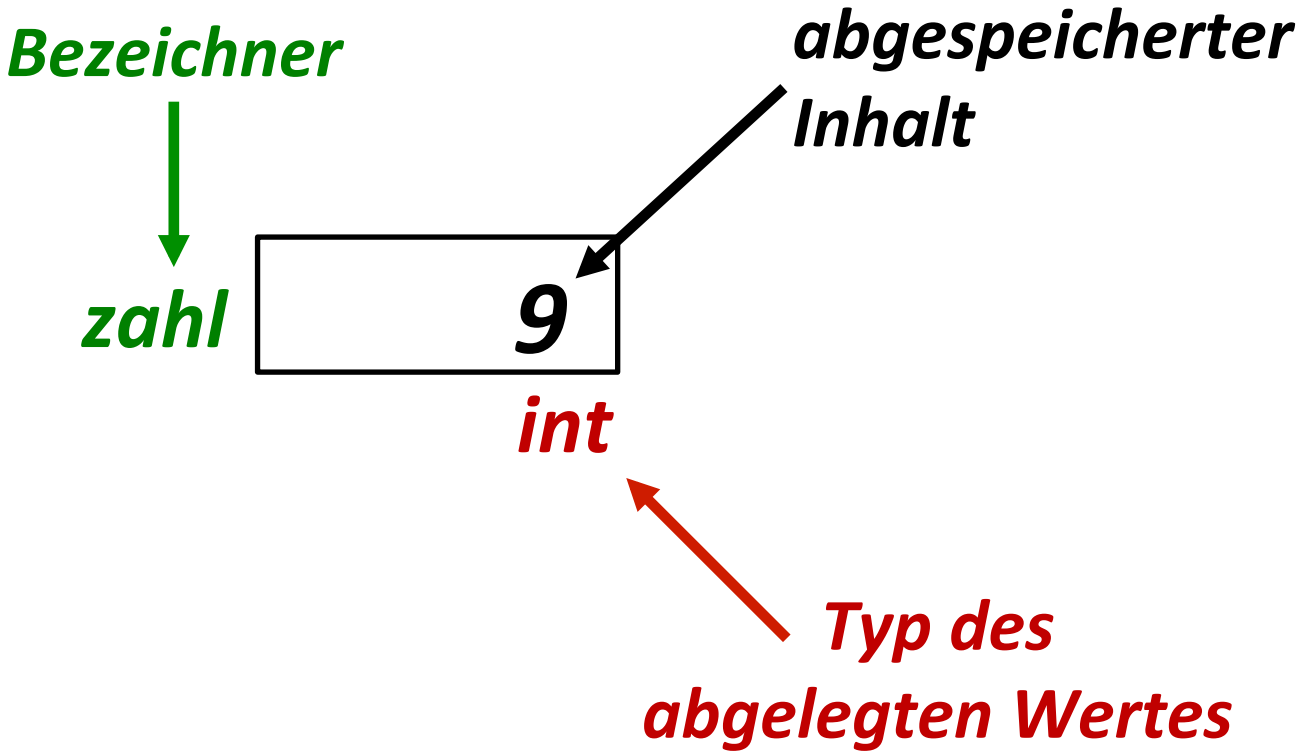
***Die an dieser Stelle gespeicherte Information wird der Wert der Variablen genannt.***

## *Erzeugen einer Variablen ……*




*Zuweisung immer von rechts nach links*

## Variablen im Speicher




*Der Typ gibt an viele Bytes zum Ablegen eines Wertes zur Verfügung stehen.*

## *Getrennte Deklaration und Initialisierung ….*

```
int zahl;
```
← ***Deklaration***

```
zahl = 8;
```
← ***Initialisierung***

```
printf("%d \n", zahl);
```

***Auslesen der Speicherstelle***

# *Ablegen von Informationen ….*

## Zusammenhang zwischen Bytes und Zahlenbereich

**Daten werden in binärer Form abgespeichert.**

**Diese binäre Form kennt zwei Zustände 0 oder 1.**

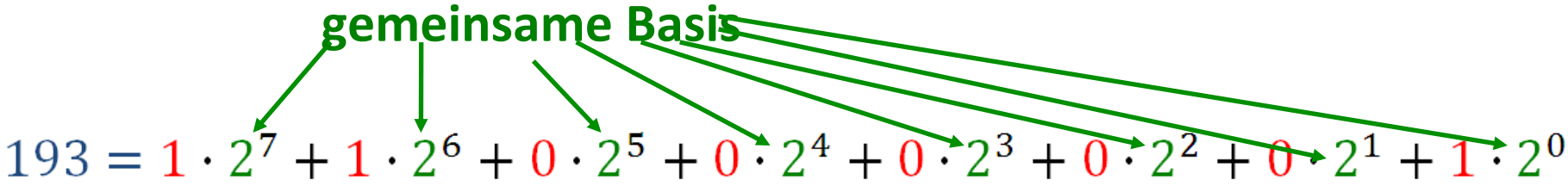
**Die kleinste Speichereinheit  nennt man 1 Bit.**

**Daten werden auf der Grundlage von Bits dargestellt.**

**8 Bit werden zu einem Byte zusammengefasst.**

## *Ablegen von Informationen ….*

**Beispiel: Wie wird die Dezimalzahl 193 binär dargestellt?**




**Vorfaktoren entsprechen der binären Darstellung:**

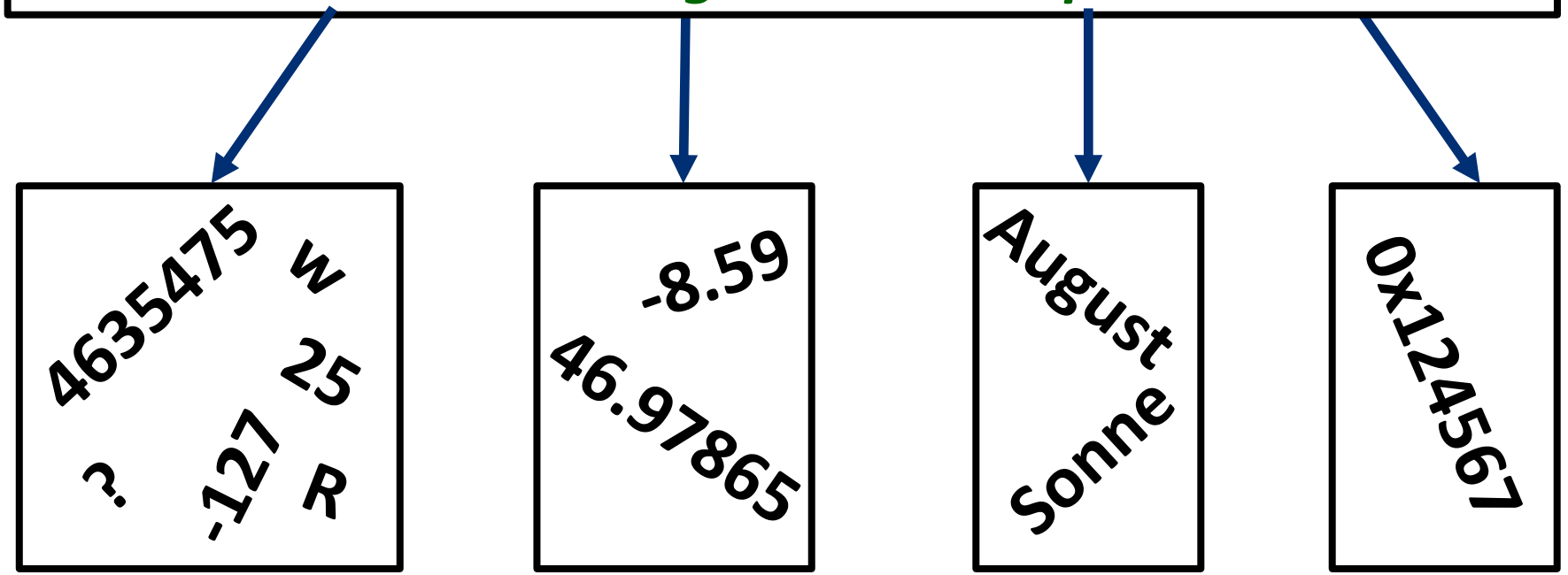
**193d entspricht 11000001b**

**Für die Ganzzahl 193 wird 1 Byte benötigt.**

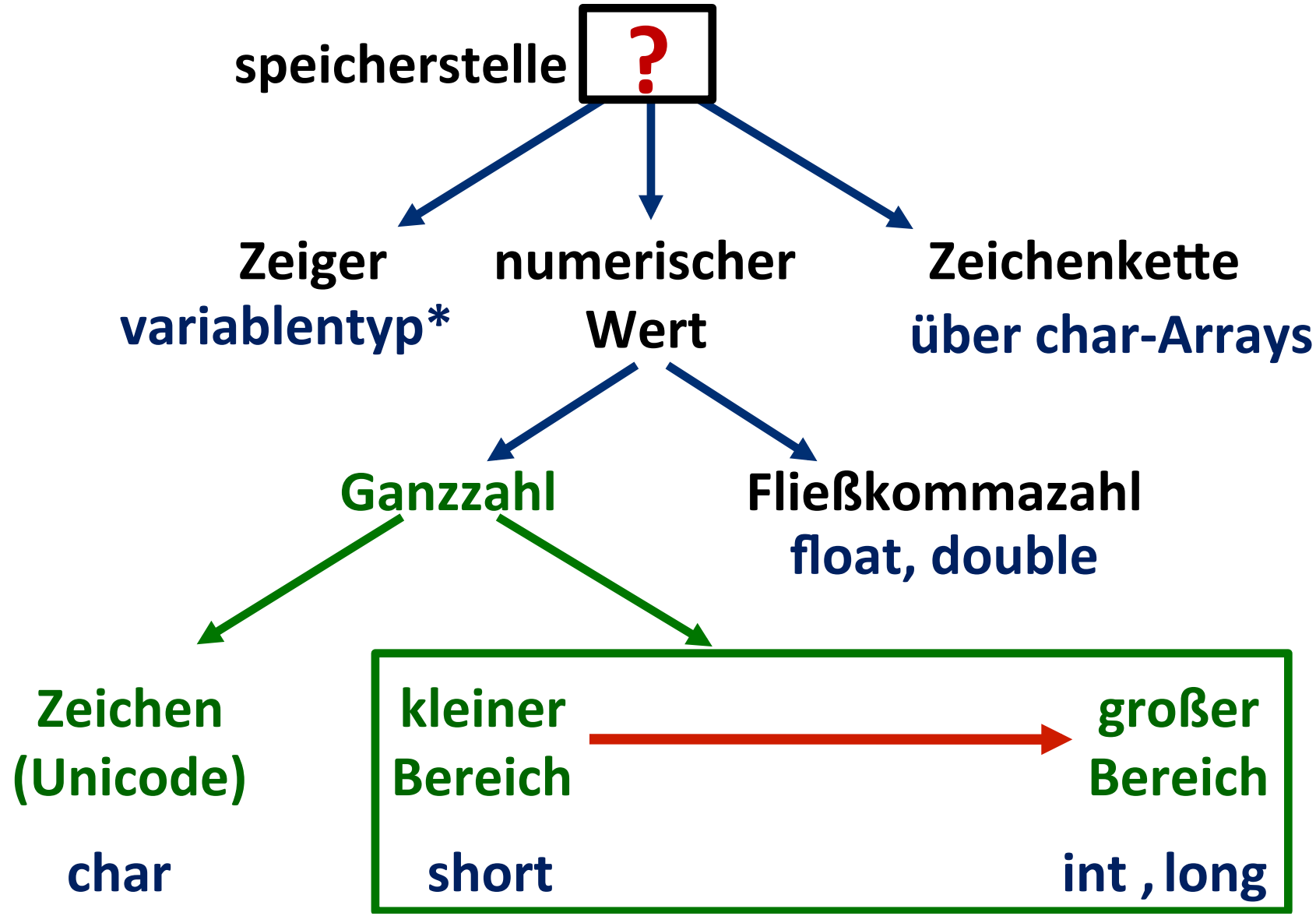
## *Warum die Angabe eines Variablentyps ....*

**speicherstelle** 0100 1001 0110 1110 .....

***Wie soll die Ansammlung an Bits interpretiert werden?***



## *Warum die Angabe eines Variablentyps ….*



# Bereich 1: Ganzzahlen (Integer)

## Integertypen

| | |
|---|---|
| char<br>unsigned char<br>signed char | short int<br>unsigned short int<br>signed short int |
| int<br>unsigned int<br>signed int | long int<br>unsigned long int<br>signed long int |

## Zusammenhang zwischen Bytes und Zahlenbereich

**char**

**1 Byte 8 Bit $2^8$ Möglichkeiten 0 … 255**

**short oder short int**

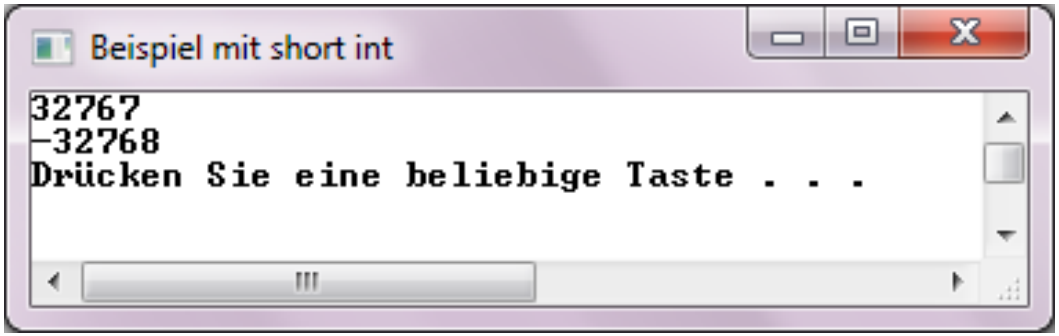
**2 Byte 16 Bit $2^{16}$ Möglichkeiten 0 … 65535**

**int oder long int**

**4 Byte 32 Bit $2^{32}$ Möglichkeiten 0 … 4294967295**

## Beispiel: Demonstration am Variablentyp short

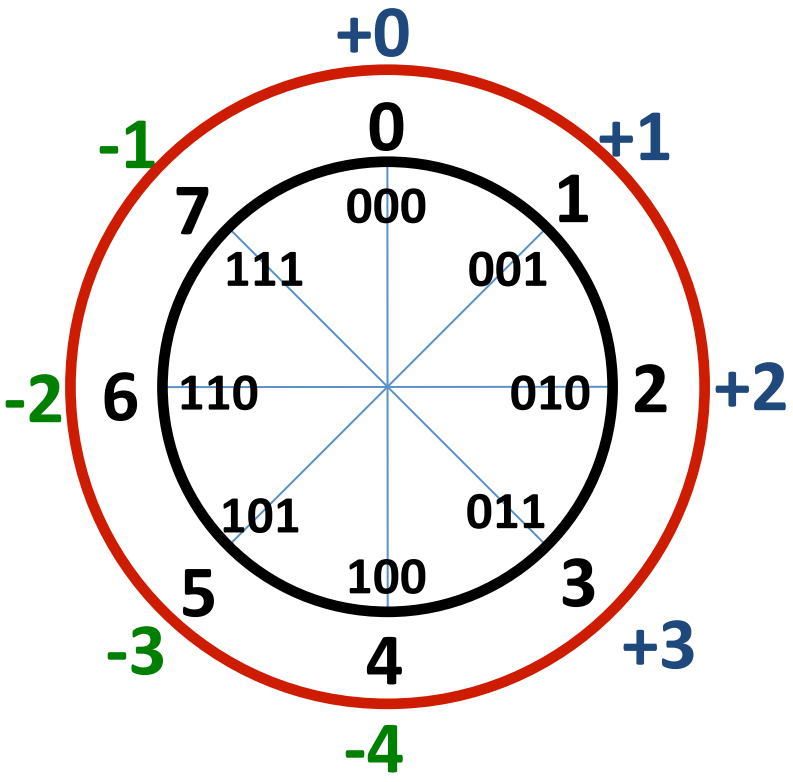
```
short zahl = 32767;

printf("%d \n", zahl);

zahl = zahl + 1;

printf("%d \n", zahl);
```




# Wie werden negative Dezimalzahlen binär dargestellt?

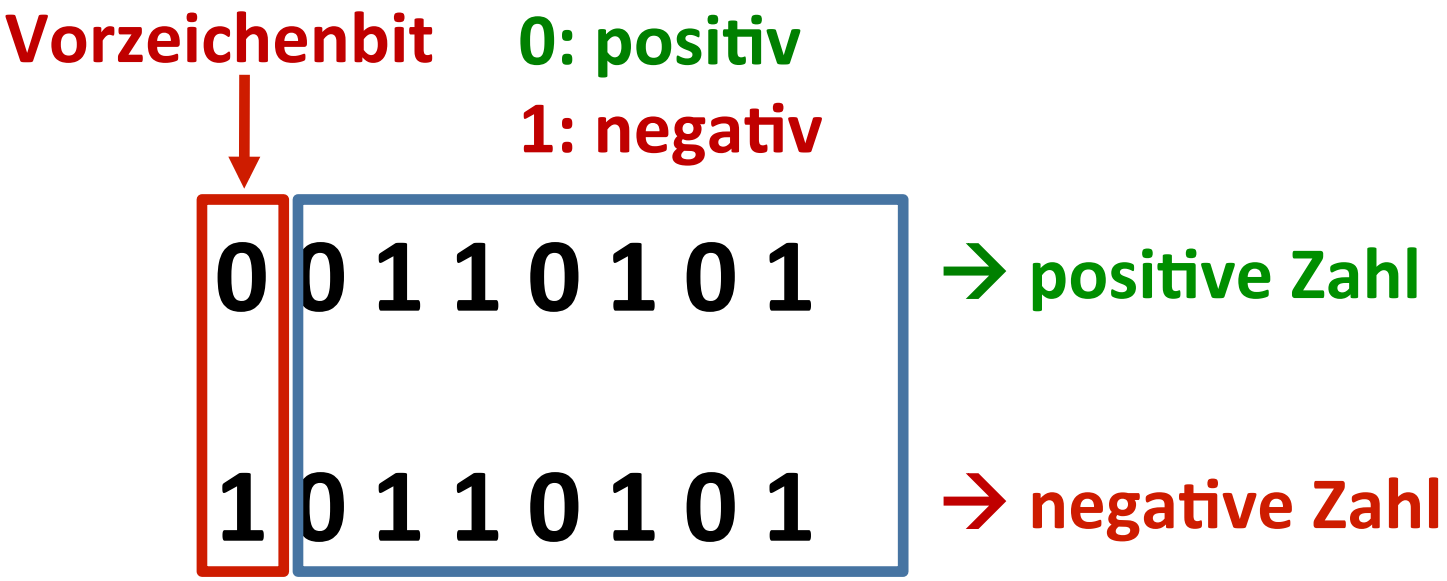
**Einfaches Beispiel: 3-Bit-Zahl:**

**3 Bit → 8 Möglichkeiten**




| | binär | m. Vz. | o. Vz. |
|---|---|---|---|
| | 000 | +0 | +0 |
| | 001 | +1 | +1 |
| | 010 | +2 | +2 |
| | 011 | +3 | +3 |
| neg. | 100 | -4 | +4 |
| neg. | 101 | -3 | +5 |
| neg. | 110 | -2 | +6 |
| neg. | 111 | -1 | +7 |

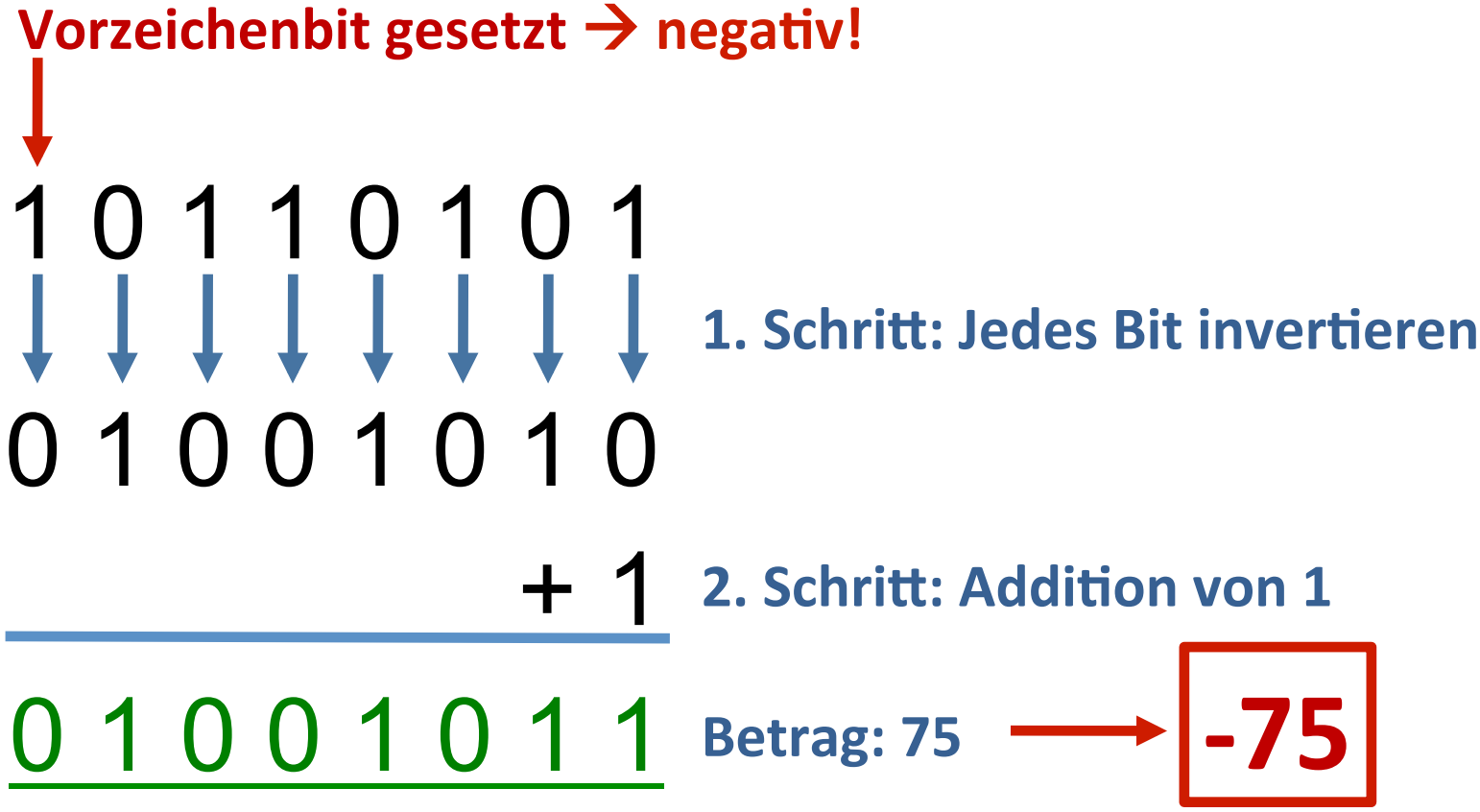
## Wie kennzeichnet man Binärzahlen als positiv oder negativ?




Nur 7 Bit zur Bildung des Betrages:

**Zahlenbereich reduziert sich: – 128 ... + 127**

## Wie bildet man von einer negativen Binärzahl den Betrag?



**Kleine Übung:**

**Bilden Sie den dezimalen Wert von den folgenden Bytezahlen:**

1 1 1 1 0 0 0 0 → - 16

0 1 1 0 0 1 1 0 → + 102

1 1 1 1 1 1 1 1 → - 1

## Klärung der Ausgabe


```
short zahl = 32767;
printf("%d \n", zahl);
zahl = zahl + 1;
printf("%d \n", zahl);
```

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

+ 1

negative Binärzahl

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**nach Invertieren und +1 gelangt man zum angezeigten Wert.**

**Konsequenz:**
**short berücksichtigt negative Ganzzahlen**

**Es kam in diesem Fall zu einem Bereichsüberlauf!!!**

# Sortieren: Integertypen in C

| | | |
|---|---|---|
| char<br>signed char | 1 Byte | - 128 … + 127 |
| unsigned char | | 0 … + 255 |

| | | |
|---|---|---|
| short oder short int<br>signed short [int] | 2 Byte | - 32768 … + 32767 |
| unsigned short [int] | | 0 … + 65535 |

| | | |
|---|---|---|
| long int, long oder int<br>signed [int, long oder long int] | 4 Byte | -2147483648 … 2147483647 |
| unsigned [int, long oder long int] | | 0 … 4294967295 |

# Welche Operationen gibt es?

## Addition:

```
int x = 17, y = 13, z = 0;

z = x + y;

printf("%d + %d = %d\n",x, y, z);
```

## Ausgabe:

```
17 + 13 = 30
```

# Welche Operationen gibt es?

## Subtraktion:

```
int x = 17, y = 13, z = 0;

z = x - y;

printf("%d - %d = %d\n",x, y, z);
```

## Ausgabe:

```
17 - 13 = 4
```

# Welche Operationen gibt es?

## Multiplikation:

```
int x = 17, y = 13, z = 0;

z = x * y;

printf("%d * %d = %d\n",x, y, z);
```

## Ausgabe:

```
17 * 13 = 221
```

## Welche Operationen gibt es?

### Division:

```
int x = 17, y = 13, z = 0;

z = x / y;

printf("%d / %d = %d\n",x, y, z);
```

### Ausgabe:

```
17 / 13 = 1
```

# Welche Operationen gibt es?

## Modulo:

```
int x = 17, y = 13, z = 0;

z = x % y;

printf("%d %% %d = %d\n",x, y, z);
```

## Ausgabe:

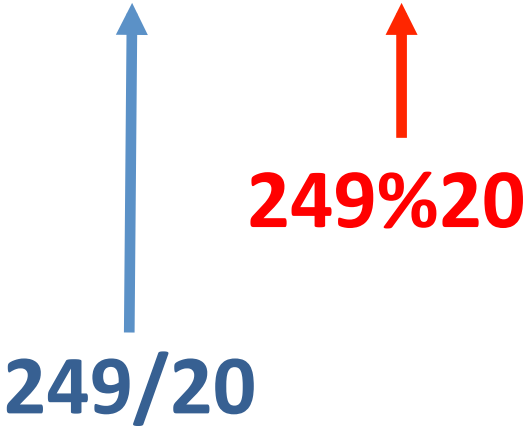
```
17 % 13 = 4
```

# Welche Operationen gibt es?

## Noch einmal Modulo-Operation:

### Ganzzahlige Berechnung:

**249 geteilt durch 20 = 12 Rest 9**



## Kleine Übung:

**Welches Ergebnis liefern die folgenden Ausdrücke?**

**a.) 17%11** → **6**

**b.) 127%128** → **127**

**c.) 360%300** → **60**

**d.) 1024%256** → **0**

## Kurzschreibweisen:

```
int x = 14;

x += 7;   // Kurzschreibweise für x = x + 7;
x -= 7;   // Kurzschreibweise für x = x - 7;
x *= 7;   // Kurzschreibweise für x = x * 7;
x /= 7;   // Kurzschreibweise für x = x / 7;

printf("%d",x);
```

## Ausgabe:

```
14
```

## Inkrementoperatoren:

## Begründen Sie die Ausgabe:

```
int x = 5;

printf("%d\n",x++ * ++x);
```


## Ausgabe:

**36**

## Inkrementoperatoren:

**Welche Aktion steht hinter diesem Befehl?**

```
x++;     // Kurzschreibweise für x = x + 1;
```

**noch ein Beispiel:**

```
int x = 2;

printf("%d, %d\n",++x, x++);
```

## Ausgabe:

**4, 2**

# Inkrementoperatoren:

**Welche Aktion steht hinter diesem Befehl?**

**Erhöhen des Variableninhaltes**

**Auslesen des Variableninhaltes**

**Möglichkeit 1:** **Zuerst Erhöhen und dann Auslesen**

**Präimkrementation** **++x**

**Möglichkeit 2:** **Zuerst Auslesen und dann Erhöhen**

**Postimkrementation** **x++**

## Inkrementoperatoren:

### Präimkrementation

### Erklärung:

Erhöhe den Wert zuerst auf 4

Lies den Wert aus (4)

```
int x = 2;

printf("%d, %d\n",++x, x++);
```

Auswertung

rechts nach links

Lies den Wert aus (2)

Erhöhe den Wert danach auf 3

### Postimkrementation

## Ausgabe:

**4, 2**

## Rangfolge der Operatoren:

```
int x = 5;

printf("%d\n",x++ * ++x);
```

**x++ wirkt auf eine einzelne Variable,**
**→ unärer Operator**

**eine Multiplikation wirkt auf zwei Variablen,**
**→ binärer Operator**

**In einem gemeinsamen Ausdruck werden unäre Operationen immer vor binären Operatoren ausgewertet.**

## Rangfolge der Operatoren:

```
int x = 5;

printf("%d\n",x++ * ++x);
```

Auswertung

rechts nach links

Lies den Wert aus (6)

Erhöhe den Wert auf 6
und lies ihn dann aus

x kann in einem Ausdruck nur einen Wert besitzen → 6

6 * 6 = 36

## Ausgabe:

36

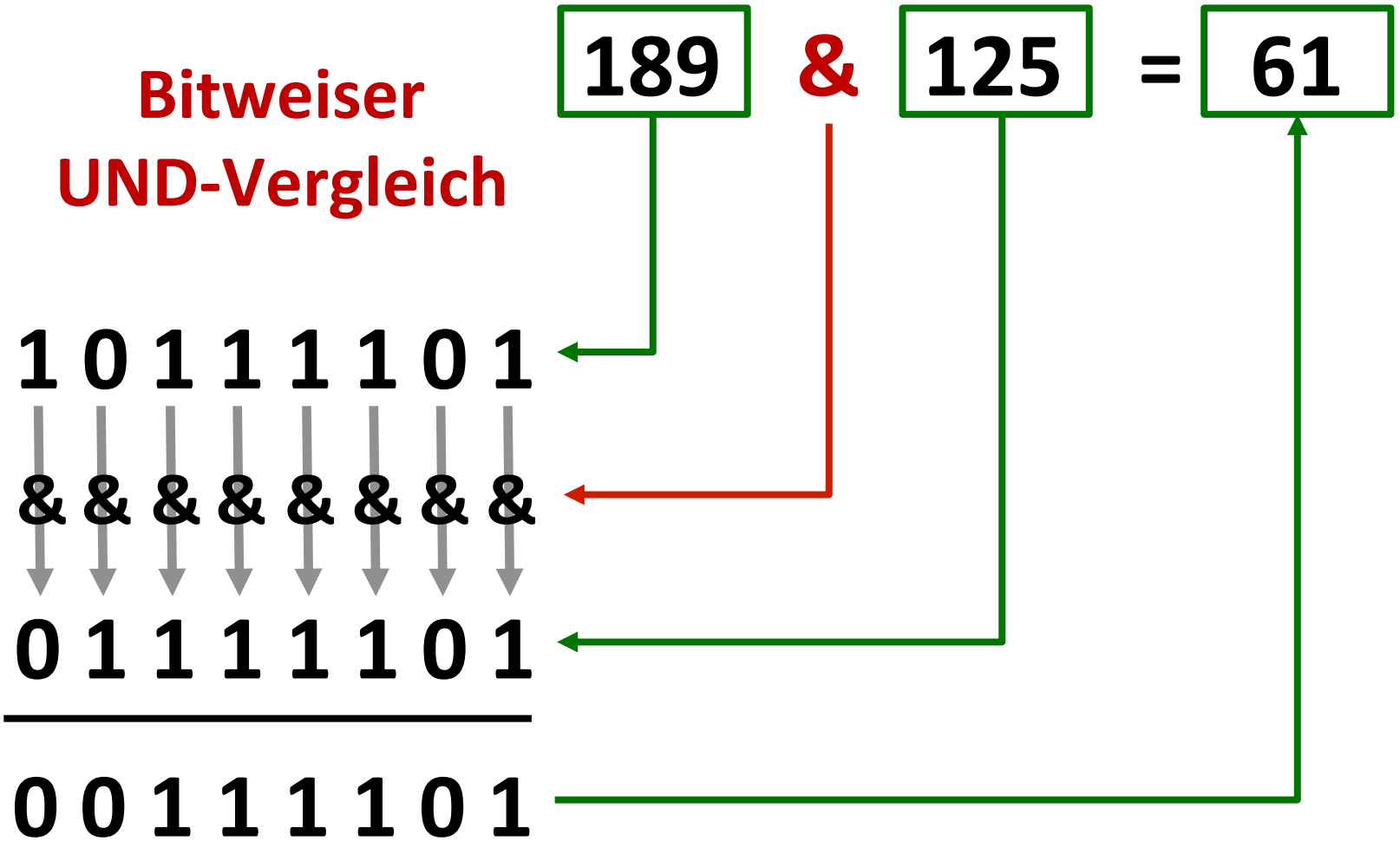
# Bitweise Operationen in C

## Erklären Sie die Ausgabe:

```
int zahl1 = 189, zahl2 = 125;

printf("%d\n",zahl1 & zahl2);

zahl1 = 192;
zahl2 = 128;

printf("%d\n",zahl1 & zahl2);

zahl1 = 255;
zahl2 = 240;

printf("%d\n",zahl1 & zahl2);
```



## Erklärung über die binäre Darstellung:




**Positionsgleiche Bits werden mit dem &-Operator verknüpft.**

## Einsatz des binären ODER-Operators:

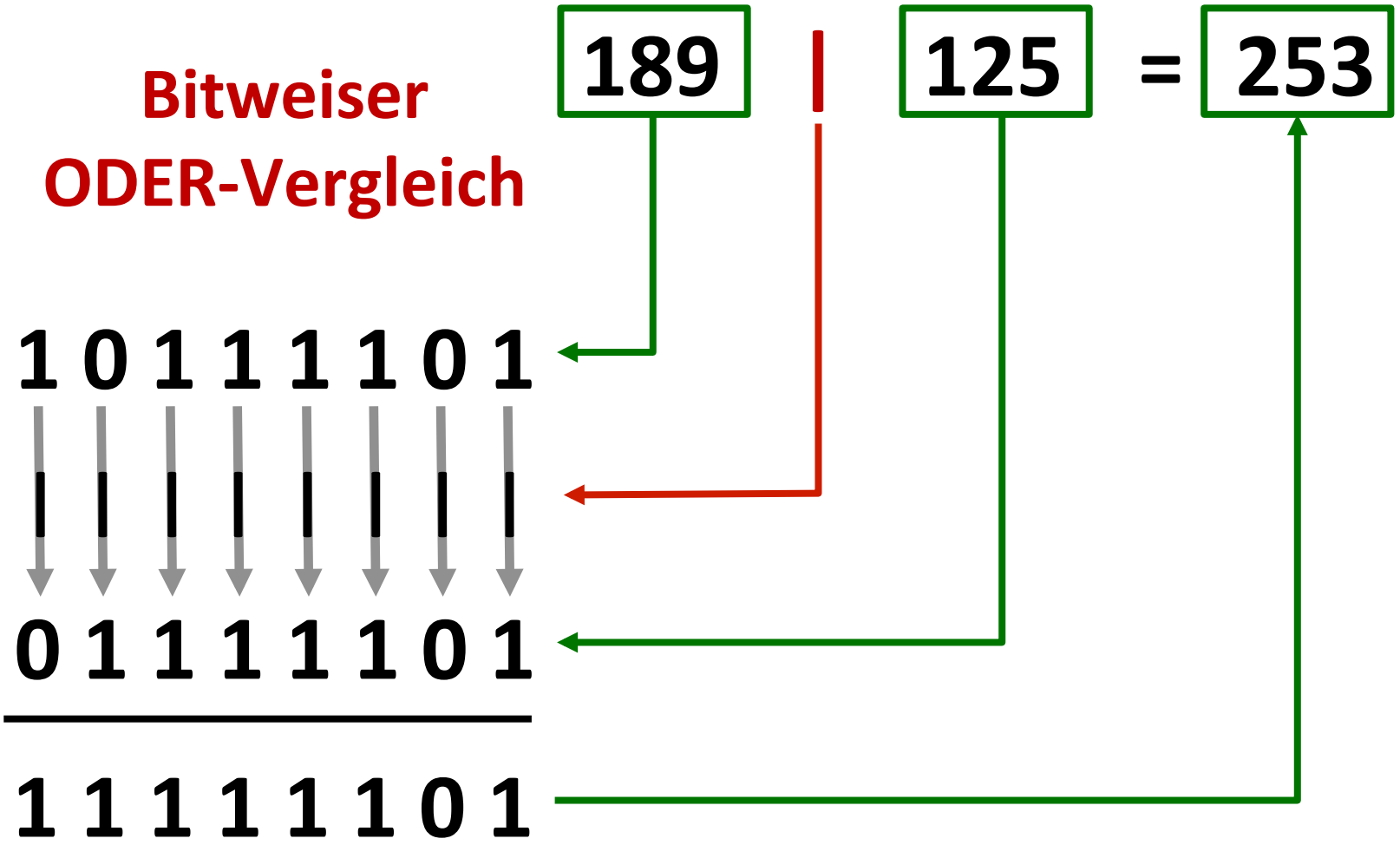
```
int zahl1 = 189, zahl2 = 125;

printf("%d\n",zahl1 | zahl2);

zahl1 = 192;
zahl2 = 128;

printf("%d\n",zahl1 | zahl2);

zahl1 = 255;
zahl2 = 240;

printf("%d\n",zahl1 | zahl2);
```

→ 253

→ 192

→ 255

## Erklärung über die binäre Darstellung:



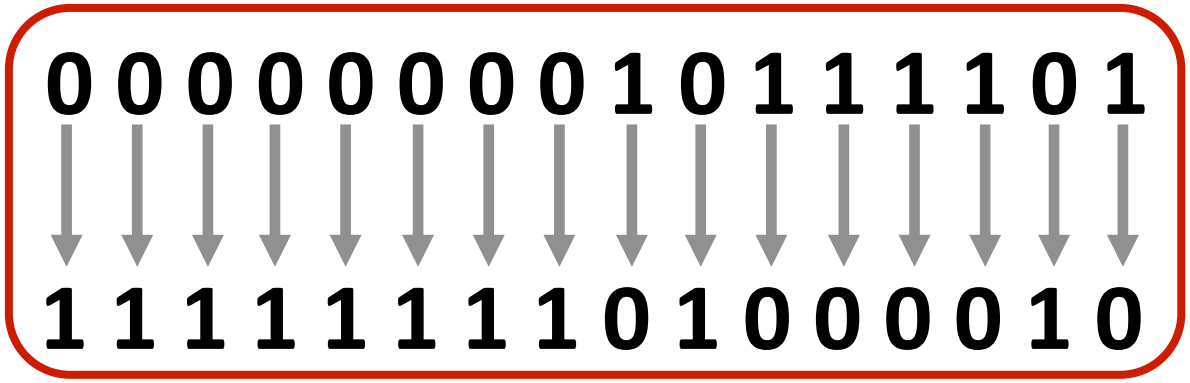

**Positionsgleiche Bits werden mit dem |-Operator verknüpft.**

## Bitweise Negation: ~

```
short zahl1 = 189;

printf("%d\n",zahl1);

zahl1 = ~zahl1;

printf("%d\n",zahl1);
```

**aus 189 wird - 190**

**warum?**




**short int**
**16 Bit**

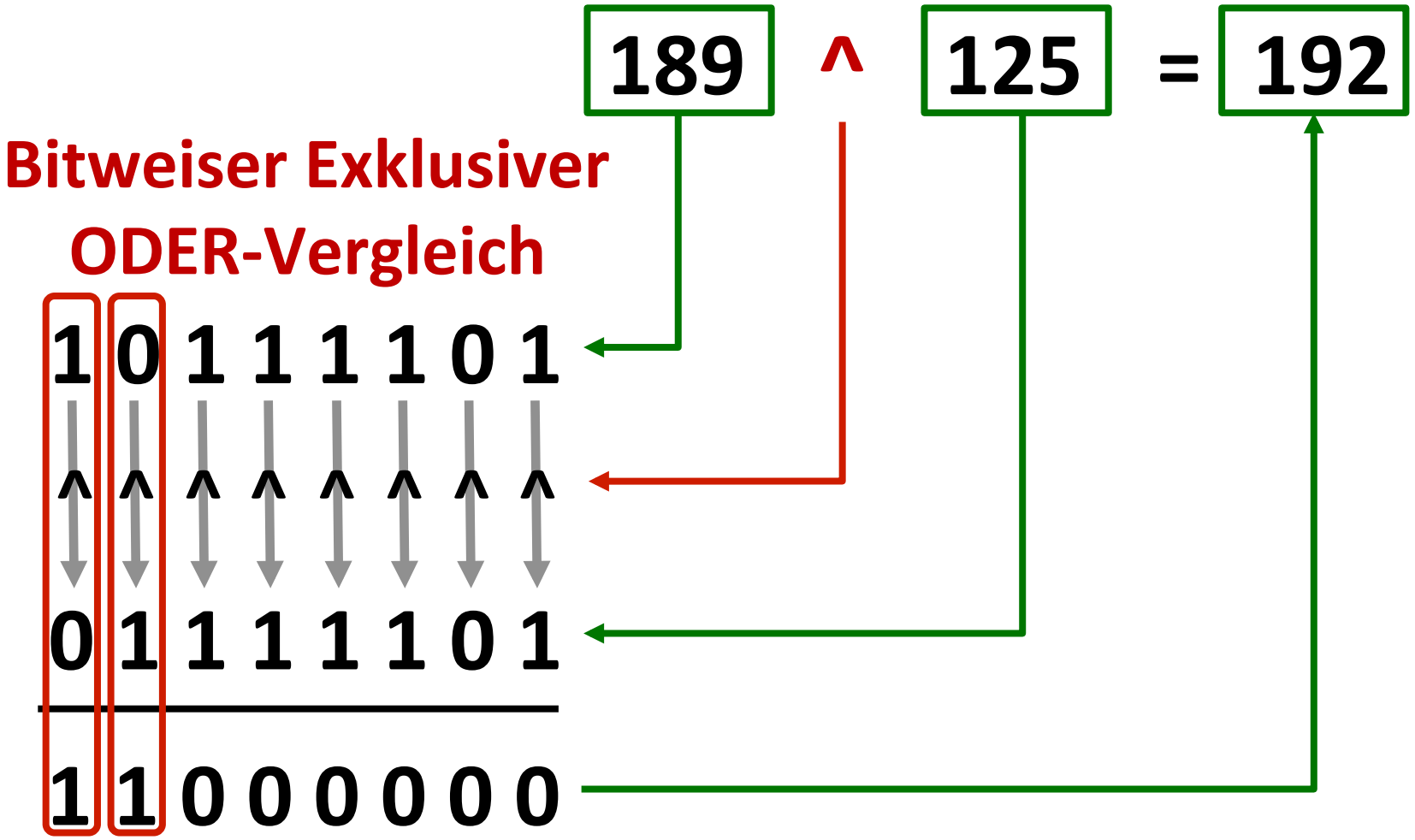
## Bitweises Exclusiv-ODER: ^ (a & ~b) | ( ~a & b)

```
short zahl1 = 189, zahl2 = 125;

printf("%d\n",zahl1^zahl2);

zahl1 = 192;
zahl2 = 128;

printf("%d\n",zahl1^zahl2);
```

**Ausgabe:** 192

64

## Erklärung über die binäre Darstellung:




XOR-Vergleich liefert „1“, wenn die Eingänge ungleich sind.

## Bitweise Verschiebe-Operatoren: <<, >>

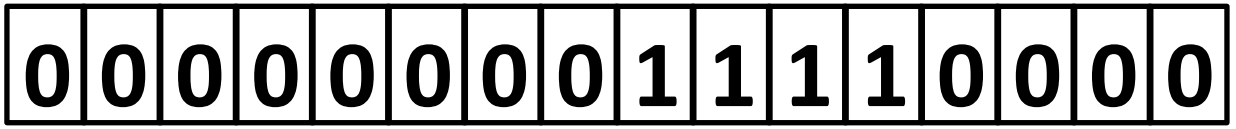
```
short zahl1 = 240;

printf("%d\n",zahl1);
```
→ 240

```
zahl1 <<= 3;

printf("%d\n",zahl1);
```
→ 1920

```
zahl1 <<= 3;

printf("%d\n",zahl1);
```
→ 15360

## Erklärung über die binäre Darstellung:

**zahl1** **189**




```
zahl1<<=3;
```

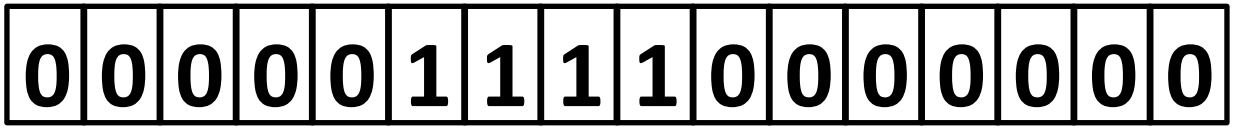
**Verschiebe um 3 binäre Stellen nach links**


**zahl1** **1920**

**Fülle mit „0“ auf**

## Erklärung über die binäre Darstellung:

**zahl1** **1920**




```
zahl1<<=3;
```

**Verschiebe um 3 binäre Stellen nach links**


**zahl1** **15360**

**Fülle mit „0“ auf**

## Bitweise Verschiebe-Operatoren: <<, >>

```
short zahl1 = 240;

printf("%d\n",zahl1);
```
→ 240

```
zahl1 >>= 3;

printf("%d\n",zahl1);
```
→ 30

```
zahl1 >>= 3;

printf("%d\n",zahl1);
```
→ 3

## Erklärung über die binäre Darstellung:

**zahl1** **189**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

```
zahl1>>=3;
```

**Verschiebe um 3 binäre Stellen nach rechts**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**zahl1** **30**

**Fülle mit „0“ auf**

## Erklärung über die binäre Darstellung:

**zahl1** **30**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

```
zahl1>>=3;
```

**Verschiebe um 3 binäre Stellen nach rechts**

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**zahl1** **3**

**Fülle mit „0“ auf**

## Bitweise Verschiebe-Operatoren: <<, >>

**Achtung bei negativen Zahlen!**

```
short zahl1 = -120;

printf("%d\n",zahl1);
```

⟶ **- 120**

```
zahl1 >>= 3;

printf("%d\n",zahl1);
```

⟶ **-15**

**Das Setzen der Füllbits ist bei negativen Zahlen compilerabhängig.**

## Erklärung über die binäre Darstellung:

**zahl1** **-120**

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

```
zahl1>>=3;
```

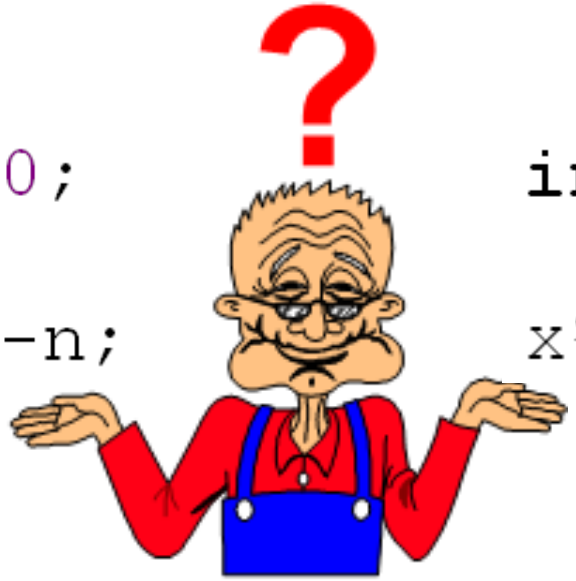
**Verschiebe um 3 binäre Stellen nach rechts**

| x | x | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**zahl1** **-15** **Hier wird mit „1“ aufgefüllt**

# *Strange oder logisch?*

```
int n=5, x=0;

x = ++n * --n;
```



```
int n=10, x=39;

x%=++n;
```

***….vielleicht helfen ein paar Übungen ….***