OPERATOREN2

Dozent: Dr. Andreas Jäger

Operatoren

- Increment, Decrement (++, --)
- Modulo (%)
- Aufsummieren, Abziehen
- Bool‘sche Logik
  - Konjunktion, Disjunktion, Negierung
- Vergleichsoperatoren

Inkrement

- Inkrement ++ (erhöhen um 1)

  ++a ist äquivalent zu a = a+1

- Decrement -- (erniedrigen um 1)

  --a ist äquivalent zu a = a-1

```
#include <iostream>
int main() {
    short i = 0;
    i++;
    std::cout << i;        //gibt 1 aus
    return 0; }
```

Inkrement

- Inkrement ++ (erhöhen um 1)

  a++ ist äquivalent zu a = a+1

- Decrement -- (erniedrigen um 1)

  a-- ist äquivalent zu a = a-1

```
#include <iostream>
int main() {
    short i = 0;
    i++;
    std::cout << i;        //gibt 1 aus
    return 0; }
```

Modulo

Division mit Rest (Modulo): %

$$a\%b = r \quad \wedge \quad a = b \times n + r \quad mit \quad 0 \leq r < b$$

117 % 5 = 2

Beispiel

```
short i = 117;
cout <<  i%5  ;        // gibt 2 auf dem Bildschirm aus
```

Ergänzung

- Aufsummieren

  langerVarName += 14;

  → Äquivalent zu:
  langerVarName = langerVarName+14;

- Subtraktionsbeispiel

  a -= b*2;

  → Äquivalent zu: **a = a –** (b*2);

Ergänzung

- a *= 2+3;

  Äquivalent zu: a = a * (2+3);
- b /= c;

  Äquivalent zu: b = b / c;
- a &= b;

  Äquivalent zu: a = a & b;
- a |= b;

  Äquivalent zu: a = a | b;

BOOL‘SCHE LOGIK

Dozent: Dr. Andreas Jäger

Ergänzung

Ausdrücke (Aussagen), die jeweils wahr bzw. falsch ergeben, können über verschiedene logische Operatoren miteinander verknüpft werden

UND, ODER, XOR

Ein Ausdruck kann auch umgekehrt (negiert) werden: NICHT

Konjunktion

Die Konjunktion zweier Ausdrücke besagt, dass bei *Ausdruck A* UND *Ausdruck B*:
Sowohl Ausdruck A als auch Ausdruck B wahr sein müssen, damit die Gesamtaussage aus beiden wahr ergibt:

UND	falsch	wahr
falsch		
wahr		

Disjunktion

A ODER B

Es reicht wenn Ausdruck A oder Ausdruck B wahr sind

UND	falsch	wahr
falsch		
wahr		

C

C kennt eigentlich keinen Datentyp für wahr bzw. falsch

Der Wert *0* entspricht der Aussage *falsch*

Alle anderen Werte bedeuten *wahr*

Erst C++ beschreibt **bool** als Datentyp mit den Werten **true** und **false**
(selbe Repräsentation, **false** entspricht dem Wert **0**)

Bitweise

- Zahlen sind binär auch aus *0*en und *1*en

- Warum nicht ermöglichen logische Verknüpfungen auch auf bit-Ebene?

→ C kennt auch bitweise logische Operationen

Konjunktion in C

- **Bitweise** Und-Verknüpfung
  - > &
    - > 1 & 2 = 0 (01 & 10)
    - > 3 & 1 = 1 (11 & 01)
    - > 3 & 3 = 3 (11 & 11)
- Logische Und-Verknüpfung
  - > &&
    - > true && 2 = true
    - > 0 && false = false

&&	0	1
0		
1		

&

- Bitweise Konjunktion

	0	1	1	0	1	0
&	1	0	1	1	1	0
=	0	0	1	0	1	0

Oder in C

- Bitweise Oder-Verknüpfung
  - > <wert> | <wert>
    - > 1 | 3 = 3 (01 | 11)
    - > 0 | 1 = 1 (00 | 01)
    - > 0 | 0 = 0 (00 | 00)
- Logische Oder-Verknüpfung
  - > <logischer Ausdruck> || <logischer Ausdruck>
    - > 1 || 2 = true
    - > false || true = true
    - > false || 0 = false

|

- Bitweises oder

	0	1	1	0	1	0
|	1	0	1	1	1	0
=	1	1	1	1	1	0

Exclusive OR

- Bitweises XOR
  - > <wert> ^ <wert>
    - >1 ^ 2 = 3 (01 ^ 10)
    - >3 ^ 1 = 2 (11 ^ 01)
    - >3 ^ 3 = 0 (11 ^ 11)
    - >0 ^ 0 = 0 (00 ^ 00)

Negierung, 1er Komplement

- Not, Nicht, Negierung
- !<logischer Ausdruck>
  - **>** ! true ist false
  - **>** ! false ist true

- Komplement (bitweises Negieren)
  - **>** ~<wert>
  - **>** ~010 = 101

Hinweis Logische Operatoren

Logische Operatoren werden nur so weit

aufgelöst, bis das Ergebnis bekannt ist:

- Bsp.: (Ausdruck a) && (Ausdruck b)
  - Wenn Ausdruck a schon falsch, dann wird Ausdruck b gar nicht erst ausgewertet.
  - Kann (ungewollte) Nebenwirkungen haben
    - z.B. nicht durchgeführter Funktionsaufruf im 2. Ausdruck.

Vergleichsoperatoren

==	Gleich, „ist a gleich b?“	(a==b)
!=	Ungleich (≠)	(a!=b)
>	Größer als	(a > b)
<	Kleiner als	(a < b)
>=	Größer oder gleich (≥)	(a >= b)
<=	Kleiner oder gleich(≤)	(a <= b)

Übung

XOR

	0	1	1	0	1	0
^	1	0	1	1	1	0
=						

Logische Ausdrücke
Ablaufplan im Pseudocode-Stil

a) Es soll eine Fehlermeldung ausgegeben werden, wenn Parameter **a** größer 15 ist oder Parameter **b** kleiner 0.

b) Ein gegebenes ***x*** muss größer als 5 und keiner als 25 sein, wenn nicht soll eine Fehlermeldung ausgeben werden.

Anmerkung

- Niemals zwei Zeichenketten (char-arrays) direkt mit Vergleichsoperatoren vergleichen

  → liefert nicht das gewünschte Ergebnis!

  siehe: strcmp, strncmp, strlen, …

  in Bibliothek <string.h>