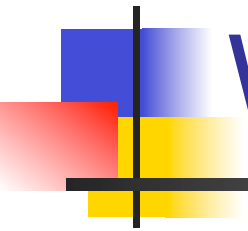
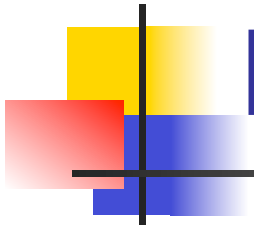


Wahrheitswerte

Boole’sche Funktionen; der Typ `bool`; logische und relationale Operatoren; Kurzschlussauswertung



Boole’sche Funktionen

- Boole’sche Funktion:

  $f: \{0,1\}^n \rightarrow \{0,1\}$

- 0 enstpricht “falsch”
- 1 entspricht “wahr”

Boole'sche Funktionen

- Boole'sche Funktion:

  $f: \{0,1\}^2 \rightarrow \{0,1\}$

- 0 enstpricht "falsch"
- 1 entspricht "wahr"

x	y	AND (x,y)
0	0	0
0	1	0
1	0	0
1	1	1

Boole'sche Funktionen

- Boole'sche Funktion:

  $f: \{0,1\}^2 \rightarrow \{0,1\}$

- 0 enstpricht "falsch"
- 1 entspricht "wahr"

x	y	OR (x,y)
0	0	0
0	1	1
1	0	1
1	1	1

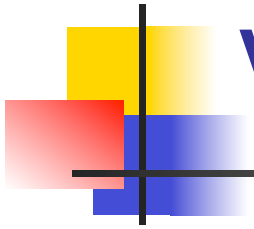
Boole’sche Funktionen

- Boole’sche Funktion:

  $f: \{0,1\}^1 \rightarrow \{0,1\}$

- 0 enstpricht “falsch”
- 1 entspricht “wahr”

x	NOT (x)
0	1
1	0



Vollständigkeit

- AND, OR, NOT sind die in C++ verfügbaren Boole'schen Funktionen
- Alle anderen *binären* boole'schen Funktionen sind daraus erzeugbar

Vollständigkeit

- AND, OR, NOT sind die in C++ verfügbaren Boole'schen Funktionen
- Alle anderen *binären* boole'schen Funktionen sind daraus erzeugbar

x	y	XOR (x,y)
0	0	0
0	1	1
1	0	1
1	1	0

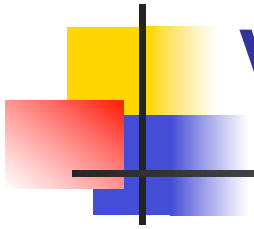
Vollständigkeit

- AND, OR, NOT sind die in C++ verfügbaren Boole'schen Funktionen
- Alle anderen *binären* boole'schen Funktionen sind daraus erzeugbar

x	y	XOR (x,y)
0	0	0
0	1	1
1	0	1
1	1	0

XOR (x,y) =

AND (OR (x,y), NOT (AND (x,y)))



Vollständigkeit: Beweis

- Identifiziere binäre boole'sche Funktion mit ihrem *charakteristischem Vektor*:

Vollständigkeit: Beweis

- Identifiziere binäre boole'sche Funktion mit ihrem *charakteristischem Vektor*:

x	y	XOR (x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Charakteristischer Vektor: 0110

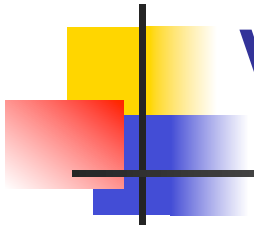
Vollständigkeit: Beweis

- Identifiziere binäre boole'sche Funktion mit ihrem *charakteristischem Vektor*:

x	y	XOR (x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Charakteristischer Vektor: 0110

XOR = f_{0110}



Vollständigkeit: Beweis

- Schritt 1: erzeuge die Funktionen f_{0001}, f_{0010}, f_{0100}, f_{1000}

Vollständigkeit: Beweis

- Schritt 1: erzeuge die Funktionen f_{0001}, f_{0010}, f_{0100}, f_{1000}

$$f_{0001} = \text{AND}\ (x, y)$$

Vollständigkeit: Beweis

- Schritt 1: erzeuge die Funktionen f_{0001}, f_{0010}, f_{0100}, f_{1000}

$$f_{0001} = \text{AND}\ (x, y)$$
$$f_{0010} = \text{AND}\ (x, \text{NOT}\ (y))$$

Vollständigkeit: Beweis

- Schritt 1: erzeuge die Funktionen f_{0001}, f_{0010}, f_{0100}, f_{1000}

$$f_{0001} = \text{AND}\ (x, y)$$
$$f_{0010} = \text{AND}\ (x, \text{NOT}\ (y))$$
$$f_{0100} = \text{AND}\ (y, \text{NOT}\ (x))$$

Vollständigkeit: Beweis

- Schritt 1: erzeuge die Funktionen f_{0001}, f_{0010}, f_{0100}, f_{1000}

f_{0001} = AND (x, y)

f_{0010} = AND (x, NOT (y))

f_{0100} = AND (y, NOT (x))

f_{1000} = NOT (OR (x, y))

Vollständigkeit: Beweis

- Schritt 2: erzeuge alle Funktionen

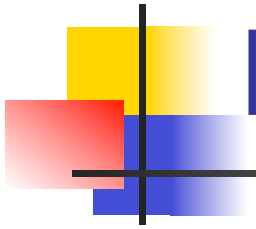
$$f_{1101} = \mathrm{OR}\,(f_{1000}, \mathrm{OR}\,(f_{0100}, f_{0001}))$$

Vollständigkeit: Beweis

- Schritt 2: erzeuge alle Funktionen

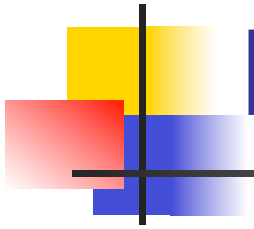
$$f_{1101} = \mathrm{OR}\,(f_{1000}, \mathrm{OR}\,(f_{0100}, f_{0001}))$$

$$f_{0000} = 0$$



Der Typ `bool`

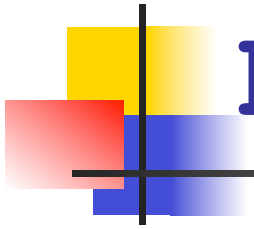
- Repräsentiert Wahrheitswerte
- Literale ***`true`*** und ***`false`***
- Wertebereich {*true, false*}



Der Typ `bool`

- Repräsentiert Wahrheitswerte
- Literale ***`true`*** und ***`false`***
- Wertebereich {*true, false*}

```
bool b = true; // Variable mit Wert true
```



`bool` vs. `int`: Konversion

- **`bool`**'s können (leider) überall dort verwendet werden, wo **`int`**'s gefordert sind, und umgekehrt

`bool` vs. `int`: Konversion

- **`bool`**'s können (leider) überall dort verwendet werden, wo **`int`**'s gefordert sind, und umgekehrt

`bool`	→	`int`
true	→	1
false	→	0

`bool` vs. `int`: Konversion

- **`bool`**'s können (leider) überall dort verwendet werden, wo **`int`**'s gefordert sind, und umgekehrt

`bool`	→	`int`
true	→	1
false	→	0

`int`	→	`bool`
≠ 0	→	*true*
0	→	*false*

`bool` vs. `int`: Konversion

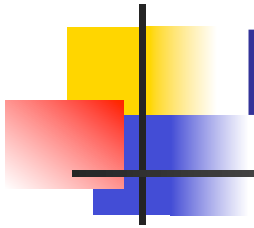
- Existierende Programme verwenden oft keine **`bool`**'s, sondern nur die **`int`**'s 0 und 1; das ist aber schlechter Stil.

`bool`	→	`int`
true	→	1
false	→	0

`int`	→	`bool`
≠ 0	→	*true*
0	→	*false*

Logische Operatoren

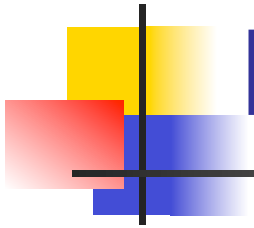
	Symbol	Stelligkeit	Präzedenz	Assoziativität
Logisches Und (AND)	&&	2	6	links
Logisches Oder (OR)	||	2	5	links
Logisches Nicht (NOT)	!	1	16	rechts



Logische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Logisches Und (AND)	&&	2	6	links
Logisches Oder (OR)	||	2	5	links
Logisches Nicht (NOT)	!	1	16	rechts

```
bool (× bool) -> bool
```

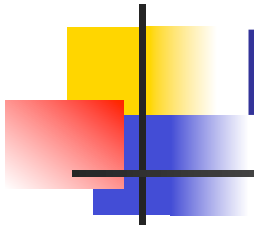


Logische Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Logisches Und (AND)	&&	2	6	links
Logisches Oder (OR)	||	2	5	links
Logisches Nicht (NOT)	!	1	16	rechts

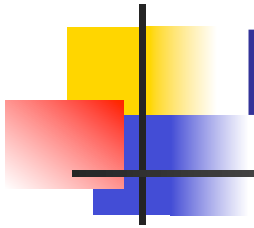
```
bool (× bool) -> bool
```

R-Wert (× R-Wert) -> R-Wert



Relationale Operatoren

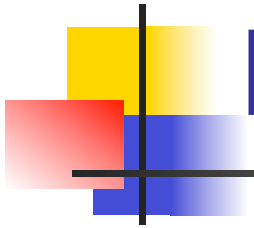
	Symbol	Stelligkeit	Präzedenz	Assoziativität
Kleiner	<	2	11	links
Grösser	>	2	11	links
Kleiner oder gleich	<=	2	11	links
Grösser oder gleich	>=	2	11	links
Gleich	==	2	10	links
Ungleich	!=	2	10	links



Relationale Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Kleiner	<	2	11	links
Grösser	>	2	11	links
Kleiner oder gleich	<=	2	11	links
Grösser oder gleich	>=	2	11	links
Gleich	==	2	10	links
Ungleich	!=	2	10	links

Zahlentyp × Zahlentyp -> **`bool`**

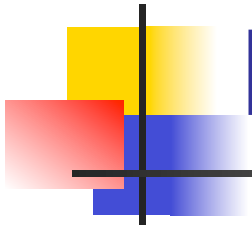


Relationale Operatoren

	Symbol	Stelligkeit	Präzedenz	Assoziativität
Kleiner	<	2	11	links
Grösser	>	2	11	links
Kleiner oder gleich	<=	2	11	links
Grösser oder gleich	>=	2	11	links
Gleich	==	2	10	links
Ungleich	!=	2	10	links

Zahlentyp × Zahlentyp -> **`bool`**

R-Wert × R-Wert -> R-Wert



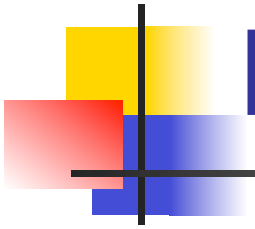
DeMorgan'sche Regeln

- `!(a && b) == (!a || !b)`
- `!(a || b) == (!a && !b)`

DeMorgan'sche Regeln

- `!(a && b) == (!a || !b)`
- `!(a || b) == (!a && !b)`

`!` (reich *und* schön) `==` (arm *oder* hässlich)



Präzedenzen

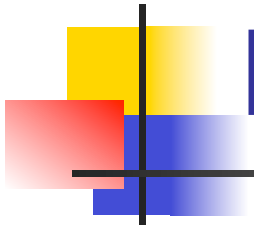
Binäre arithmetische Operatoren

binden stärker als

Relationale Operatoren,

und diese binden stärker als

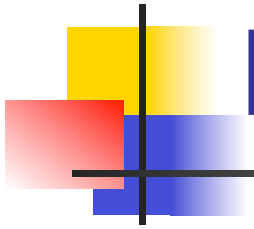
binäre logische Operatoren.



Präzedenzen

Binäre arithmetische Operatoren
binden stärker als
Relationale Operatoren,
und diese binden stärker als
binäre logische Operatoren.

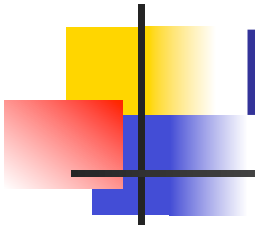
```
7 + x  <  y  &&  y  !=  3 * z
```



Präzedenzen

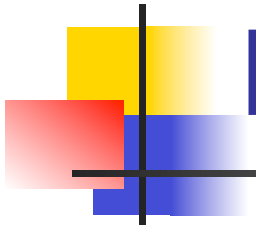
Binäre arithmetische Operatoren
binden stärker als
Relationale Operatoren,
und diese binden stärker als
binäre logische Operatoren.

```
((7 + x) <  y) && (y  != (3 * z))
```



Kurzschlussauswertung

- Logische Operatoren **&&** und **||** werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet



Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

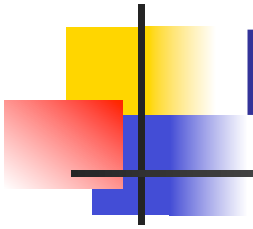
```
x != 0 && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 1:

```
x != 0 && z / x > y
```



Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 1:

```
x != 0 && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 1:

```
true    && z / x > y
```

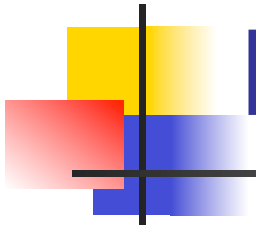
Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 1:

```
true    && z / x > y
```

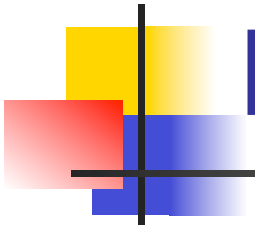
Kurzschlussauswertung



- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 0:

```
x != 0 && z / x > y
```



Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 0:

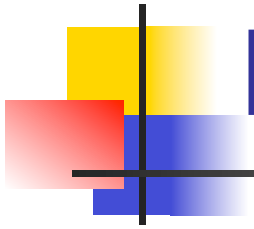
```
x != 0 && z / x > y
```

Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 0:

```
false  && z / x > y
```



Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

`x` hat Wert 0: **`false`**

Kurzschlussauswertung

- Logische Operatoren `&&` und `||` werten den *linken* Operanden zuerst aus
- Falls das Ergebnis dann schon feststeht, wird der rechte Operand *nicht mehr* ausgewertet

```
x != 0 && z / x > y
```

keine Division durch Null!