

# 1. Einführung

Informatik: Definition und Geschichte, Algorithmen, Turing Maschine, Höhere Programmiersprachen, Werkzeuge der Programmierung, das erste C++ Programm und seine syntaktischen und semantischen Bestandteile

## Was ist Informatik?

- Die Wissenschaft der **systematischen Verarbeitung von Informationen**, . . .
- . . . insbesondere der automatischen Verarbeitung mit Hilfe von Digitalrechnern.

(Wikipedia, nach dem “Duden Informatik”)

## Informatik $\neq$ Computer Science

*Computer science is not about machines, in the same way that astronomy is not about telescopes.*

Mike Fellows, US-Informatiker (1991)

## Computer Science $\subseteq$ Informatik

- Die Informatik beschäftigt sich heute auch mit dem Entwurf von schnellen Computern und Netzwerken. . .
- . . . aber nicht als Selbstzweck, sondern zur effizienteren **systematischen Verarbeitung von Informationen**.

## Informatik $\neq$ EDV-Kenntnisse

EDV-Kenntnisse: *Anwenderwissen*

- Umgang mit dem Computer
- Bedienung von Computerprogrammen (für Texterfassung, Email, Präsentationen, ...)

Informatik: *Grundlagenwissen*

- Wie funktioniert ein Computer?
- Wie schreibt man ein Computerprogramm?

28

## Inhalt dieser Vorlesung

- Systematisches Problemlösen mit Algorithmen und der Programmiersprache C++.
- Also: *nicht nur, aber auch* Programmierkurs.

29

## Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Handlungsanweisung zur schrittweisen Lösung eines Problems
- Ausführung erfordert keine Intelligenz, nur Genauigkeit (sogar Computer können es)
- nach *Muhammed al-Chwarizmi*, Autor eines arabischen Rechen-Lehrbuchs (um 825)



"Dixit algorizmi..." (lateinische Übersetzung)

<http://de.wikipedia.org/wiki/Algorithmus>

## Der älteste nichttriviale Algorithmus

Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)

- Eingabe: ganze Zahlen  $a > 0, b > 0$
- Ausgabe: ggT von  $a$  und  $b$

Solange  $b \neq 0$

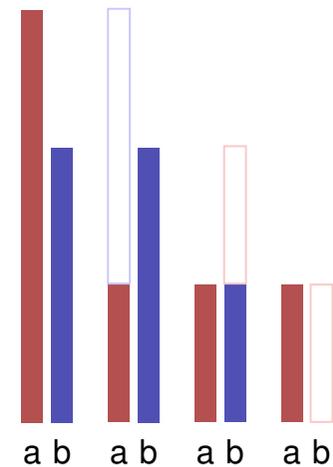
Wenn  $a > b$  dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .



31



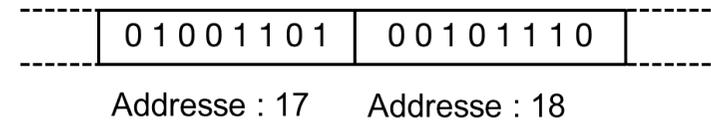
## Computer

Zutaten der *Von Neumann Architektur*:

- Hauptspeicher (RAM) für Programme *und* Daten
- Prozessor (CPU) zur Verarbeitung der Programme und Daten
- I/O Komponenten zur Kommunikation mit der Aussenwelt

## Speicher für Daten *und* Programm

- Folge von Bits aus  $\{0, 1\}$ .
- Programmzustand: Werte aller Bits.
- Zusammenfassung von Bits zu Speicherzellen (oft: 8 Bits = 1 Byte).
- Jede Speicherzelle hat eine Adresse.
- Random Access: Zugriffszeit auf Speicherzelle (nahezu) unabhängig von ihrer Adresse.



36

37

## Prozessor

Der Prozessor (CPU)

- führt Befehle in Maschinensprache aus
- hat eigenen "schnellen" Speicher (Register)
- kann vom Hauptspeicher lesen und in ihn schreiben
- beherrscht eine Menge einfachster Operationen (z.B. Addieren zweier Registerinhalte)

## Rechengeschwindigkeit

In der mittleren Zeit, die der Schall von mir zu Ihnen unterwegs ist...

30 m  $\hat{=}$  mehr als 100.000.000 Instruktionen

arbeitet ein heutiger Desktop-PC mehr als 100 Millionen Instruktionen ab.<sup>2</sup>

<sup>2</sup>Uniprozessor Computer bei 1GHz

38

39

## Programmieren

- Mit Hilfe einer *Programmiersprache* wird dem Computer eine Folge von Befehlen erteilt, damit er genau das macht, was wir wollen.
- Die Folge von Befehlen ist das *(Computer)-Programm*.



The Harvard Computers, Menschliche Berufsrechner, ca.1890

[http://en.wikipedia.org/wiki/Harvard\\_Computers](http://en.wikipedia.org/wiki/Harvard_Computers)

40

*Mathematik war früher die Lingua franca der Naturwissenschaften an allen Hochschulen. Und heute ist dies die Informatik.*

*Lino Guzzella, Präsident der ETH Zürich, NZZ Online, 1.9.2017*

42

## Warum Programmieren?

- Da hätte ich ja gleich Informatik studieren können ...
- Es gibt doch schon für alles Programme ...
- Programmieren interessiert mich nicht ...
- Weil Informatik hier leider ein Pflichtfach ist ...
- ...

41

## Darum Programmieren!

- Jedes Verständnis moderner Technologie erfordert Wissen über die grundlegende Funktionsweise eines Computers.
- Programmieren (mit dem Werkzeug Computer) wird zu einer Kulturtechnik wie Lesen und Schreiben (mit den Werkzeugen Papier und Bleistift)
- Programmieren ist *die* Schnittstelle zwischen Ingenieurwissenschaften und Informatik – der interdisziplinäre Grenzbereich wächst zusehends.
- Programmieren macht Spass!

43

## Programmiersprachen

- Sprache, die der Computer "versteht", ist sehr primitiv (Maschinensprache).
- Einfache Operationen müssen in viele Einzelschritte aufgeteilt werden.
- Sprache variiert von Computer zu Computer.

44

## Höhere Programmiersprachen

darstellbar als Programmtext, der

- von Menschen *verstanden* werden kann
- vom Computermodell *unabhängig* ist  
→ Abstraktion!

45

## Programmiersprachen – Einordnung

Unterscheidung in

- **Kompilierte** vs. interpretierte Sprachen
  - *C++*, C#, Pascal, Modula, Oberon, Java  
vs.  
Python, Tcl, Matlab
- **Höhere** Programmiersprachen vs. Assembler.
- **Mehrzweck**sprachen vs. zweckgebundene Sprachen.
- **Prozedurale, Objekt-Orientierte**, Funktionsorientierte und logische Sprachen.

46

## Warum C++?

Andere populäre höhere Programmiersprachen: Java, C#, Objective-C, Modula, Oberon, Python ...

Allgemeiner Konsens

- „Die“ Programmiersprache für Systemprogrammierung: C
- C hat erhebliche Schwächen. Grösste Schwäche: fehlende Typsicherheit.

47

## Warum C++?

*Over the years, C++'s greatest strength and its greatest weakness has been its C-Compatibility – B. Stroustrup*

B. Stroustrup, Design and Evolution of C++, Kap. 4.5  
48

## Warum C++?

- C++ versieht C mit der Mächtigkeit der Abstraktion einer höheren Programmiersprache
- In diesem Kurs: C++ als Hochsprache eingeführt (nicht als besseres C)
- Vorgehen: Traditionell prozedural → objekt-orientiert

49

## Deutsch vs. C++

### Deutsch

*Es ist nicht genug zu wissen,  
man muss auch anwenden.  
(Johann Wolfgang von Goethe)*

### C++

```
// computation
int b = a * a; // b = a^2
b = b * b;    // b = a^4
```

50

## Syntax und Semantik

- Programme müssen, wie unsere Sprache, nach gewissen Regeln geformt werden.
  - **Syntax**: Zusammenfügingsregeln für elementare Zeichen (Buchstaben).
  - **Semantik**: Interpretationsregeln für zusammengefügte Zeichen.
- Entsprechende Regeln für ein Computerprogramm sind einfacher, aber auch strenger, denn Computer sind vergleichsweise dumm.

51

## C++: Fehlerarten illustriert an deutscher Sprache

- Das Auto fuhr zu schnell.
- DasAuto fuh r zu sxhnell.
- Rot das Auto ist.
- Man empfiehlt dem Dozenten nicht zu widersprechen
- Sie ist nicht gross und rothaarig.
- Die Auto ist rot.
- Das Fahrrad gallopiert schnell.
- Manche Tiere riechen gut.

Syntaktisch und semantisch korrekt.

Syntaxfehler: Wortbildung.

Syntaxfehler: Satzstellung.

Syntaxfehler: Satzzeichen fehlen .

Syntaktisch korrekt aber mehrdeutig. [kein Analogon]

Syntaktisch korrekt, doch semantisch fehlerhaft: Falscher Artikel. [Typfehler]

Syntaktisch und grammatikalisch korrekt! Semantisch fehlerhaft. [Laufzeitfehler]

Syntaktisch und semantisch korrekt. Semantisch mehrdeutig. [kein Analogon]

52

## Syntax und Semantik von C++

### Syntax

- Was *ist* ein C++ Programm?
- Ist es *grammatikalisch* korrekt?

### Semantik

- Was *bedeutet* ein Programm?
- Welchen Algorithmus realisiert ein Programm?

53

## Syntax und Semantik von C++

Der ISO/IEC Standard 14822 (1998, 2011,...)

- ist das "Gesetz" von C++
- legt Grammatik und Bedeutung von C++-Programmen fest
- enthält seit 2011 Neuerungen für *fortgeschrittenes* Programmieren. . .
- . . . weshalb wir auf diese Neuerungen hier auch nicht weiter eingehen werden.

54

## Was braucht es zum Programmieren?

- **Editor:** Programm zum Ändern, Erfassen und Speichern von C++-Programmtext
- **Compiler:** Programm zum Übersetzen des Programmtexts in Maschinensprache
- **Computer:** Gerät zum Ausführen von Programmen in Maschinensprache
- **Betriebssystem:** Programm zur Organisation all dieser Abläufe (Dateiverwaltung, Editor-, Compiler- und Programmaufruf)

55

## Sprachbestandteile am Beispiel

- Kommentare/Layout
- Include-Direktiven
- Die main-Funktion
- Werte, Effekte
- Typen, Funktionalität
- Literale
- Variablen
- Konstanten
- Bezeichner, Namen
- Objekte
- **Ausdrücke**
- L- und R-Werte
- Operatoren
- Anweisungen

## Das erste C++ Programm Wichtigste Bestandteile...

```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
int main(){
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a; ← Anweisungen: Mache etwas (lies a ein)!
    // computation
    int b = a * a; // b = a^2 ← Ausdrücke: Berechne einen Wert (a^2)!
    b = b * b;    // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}
```

56

57

## Verhalten eines Programmes

Zur Compilationszeit:

- vom Compiler akzeptiertes Programm (syntaktisch korrektes C++)
- Compiler-Fehler

Zur Laufzeit:

- korrektes Resultat
- inkorrektes Resultat
- Programmabsturz
- Programm *terminiert* nicht (Endlosschleife)

## “Beiwerk”: Kommentare

```
// Program: power8.cpp
// Raise a number to the eighth power. ← Kommentare
#include <iostream>
int main() {
    // input ←
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation ←
    int b = a * a; // b = a^2
    b = b * b;    // b = a^4
    // output b * b, i.e., a^8 ←
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}
```

58

59

## Kommentare und Layout

### Kommentare

- hat jedes gute Programm,
- dokumentieren, *was* das Programm *wie* macht und wie man es verwendet und
- werden vom Compiler ignoriert.
- Syntax: “Doppelslash” // bis Zeilenende.

### Ignoriert werden vom Compiler ausserdem

- Leerzeilen, Leerzeichen,
- Einrückungen, die die Programmlogik widerspiegeln (sollten)

60

## Kommentare und Layout

### Dem Compiler ist's egal...

```
#include <iostream>
int main(){std::cout << "Compute a^8 for a =? ";
int a; std::cin >> a; int b = a * a; b = b * b;
std::cout << a << "^8 = " << b*b << "\n";return 0;}
```

### ... uns aber nicht!

61

## “Beiwerk”: Include und Main-Funktion

```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream> ← Include-Direktive
int main() { ← Funktionsdeklaration der main-Funktion
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation
    int b = a * a; // b = a^2
    b = b * b;    // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}
```

62

## Include-Direktiven

### C++ besteht aus

- Kernsprache
- Standardbibliothek
  - Ein/Ausgabe (Header iostream)
  - Mathematische Funktionen (cmath)
  - ...

```
#include <iostream>
```

- macht Ein/Ausgabe verfügbar

63

## Die Hauptfunktion

Die `main`-Funktion

- existiert in jedem C++ Programm
- wird vom Betriebssystem aufgerufen
- wie eine mathematische Funktion ...
  - Argumente (bei `power8.cpp`: keine)
  - Rückgabewert (bei `power8.cpp`: 0)
- ... aber mit zusätzlichem *Effekt*.
  - Lies eine Zahl ein und gib die 8-te Potenz aus.

64

## Anweisungen: Mache etwas!

```
int main() {  
    // input  
    std::cout << "Compute a^8 for a =? ";  
    int a;  
    std::cin >> a;  
    // computation  
    int b = a * a; // b = a^2  
    b = b * b; // b = a^4  
    // output b * b, i.e., a^8  
    std::cout << a << "^8 = " << b * b << "\n";  
    return 0;  
}
```

Ausdrucksanweisungen

Rückgabeanweisung

65

## Anweisungen

- Bausteine eines C++ Programms
- werden (sequenziell) *ausgeführt*
- enden mit einem Semikolon
- Jede Anweisung hat (potenziell) einen *Effekt*.

66

## Ausdrucksanweisungen

- haben die Form  
`expr;`  
wobei `expr` ein Ausdruck ist
- Effekt ist der Effekt von `expr`, der Wert von `expr` wird ignoriert.

Beispiel: `b = b*b;`

67

## Rückgabeanweisungen

- treten nur in Funktionen auf und sind von der Form

`return expr;`

wobei *expr* ein Ausdruck ist

- spezifizieren Rückgabewert der Funktion

Beispiel: `return 0;`

## Anweisungen – Effekte

```
int main() {  
    // input  
    std::cout << "Compute a^8 for a =? ";  
    int a;  
    std::cin >> a;  
    // computation  
    int b = a * a; // b = a^2  
    b = b * b; // b = a^4  
    // output b * b, i.e., a^8  
    std::cout << a << "^8 = " << b * b << "\n";  
    return 0;  
}
```

Effekt: Ausgabe des Strings Compute ...

Effekt: Eingabe einer Zahl und Speichern in a

Effekt: Speichern des berechneten Wertes von a\*a in b

Effekt: Speichern des berechneten Wertes von b\*b in b

Effekt: Rückgabe des Wertes 0

Effekt: Ausgabe des Wertes von a und des berechneten Wertes von b\*b

68

69

## Werte und Effekte

- bestimmen, was das Programm macht,
- sind rein semantische Konzepte:
  - Zeichen 0 bedeutet Wert  $0 \in \mathbb{Z}$
  - `std::cin >> a;` bedeutet Effekt "Einlesen einer Zahl"
- hängen vom Programmzustand (Speicherinhalte / Eingaben) ab

## Anweisungen – Variablendefinitionen

```
int main() {  
    // input  
    std::cout << "Compute a^8 for a =? ";  
    int a;  
    std::cin >> a;  
    // computation  
    int b = a * a; // b = a^2  
    b = b * b; // b = a^4  
    // output b * b, i.e., a^8  
    std::cout << a << "^8 = " << b * b << "\n";  
    return 0;  
}
```

Typnamen

Deklarationsanweisungen

70

71

## Deklarationsanweisungen

- führen neue Namen im Programm ein,
- bestehen aus Deklaration + Semikolon

Beispiel: `int a;`

- können Variablen auch initialisieren

Beispiel: `int b = a * a;`

72

## Typen und Funktionalität

`int`:

- C++ Typ für ganze Zahlen,
- entspricht  $(\mathbb{Z}, +, \times)$  in der Mathematik

In C++ hat jeder Typ einen Namen sowie

- Wertebereich (z.B. ganze Zahlen)
- Funktionalität (z.B. Addition/Multiplikation)

73

## Fundamentaltypen

C++ enthält fundamentale Typen für

- Ganze Zahlen (`int`)
- Natürliche Zahlen (`unsigned int`)
- Reelle Zahlen (`float`, `double`)
- Wahrheitswerte (`bool`)
- ...

74

## Literale

- repräsentieren konstante Werte,
- haben festen *Typ* und *Wert*
- sind "syntaktische Werte".

Beispiele:

- 0 hat Typ `int`, Wert 0.
- `1.2e5` hat Typ `double`, Wert  $1.2 \cdot 10^5$ .

75

## Variablen

- repräsentieren (wechselnde) Werte,
- haben
  - *Name*
  - *Typ*
  - *Wert*
  - *Adresse*
- sind im Programmtext "sichtbar".

### Beispiel

```
int a; definiert Variable mit
```

- Name: a
- Typ: int
- Wert: (vorerst) undefiniert
- Adresse: durch Compiler (und Linker, Laufzeit)bestimmt

76

## Objekte

- repräsentieren Werte im Hauptspeicher
- haben *Typ*, *Adresse* und *Wert* (Speicherinhalt an der Adresse),
- können benannt werden (Variable) ...
- ... aber auch anonym sein.

### Anmerkung

Ein Programm hat eine  *feste* Anzahl von Variablen. Um eine  *variable* Anzahl von Werten behandeln zu können, braucht es "anonyme" Adressen, die über temporäre Namen angesprochen werden können.

77

## Bezeichner und Namen

(Variablen-)Namen sind Bezeichner:

- erlaubt: A,...,Z; a,...,z; 0,...,9;\_
- erstes Zeichen ist Buchstabe.

Es gibt noch andere Namen:

- `std::cin` (qualifizierter Name)

78

## Ausdrücke: Berechne einen Wert!

- repräsentieren *Berechnungen*
- sind entweder *primär* (b)
- oder *zusammengesetzt* (b\*b)...
- ... aus anderen Ausdrücken, mit Hilfe von *Operatoren*
- haben einen Typ und einen Wert

Analogie: Baukasten

79

# Ausdrücke

# Baukasten

# Ausdrücke (Expressions)

- repräsentieren *Berechnungen*,
- sind *primär* oder *zusammengesetzt* (aus anderen Ausdrücken und Operationen)

`a * a`  
 zusammengesetzt aus  
 Variablenname, Operatorsymbol, Variablenname  
 Variablenname: primärer Ausdruck

- können geklammert werden

`a * a` ist äquivalent zu `(a * a)`

```
// input
std::cout << "Compute a^8 for a=? ";
int a;
std::cin >> a;

// computation
int b = a * a; // b = a^2
b = b * b; // b = a^4
// output b * b, i.e., a^8
std::cout << a << "^8 = " << b * b << ".\n";

return 0;
```

Zusammengesetzter Ausdruck (pointing to the first line)

Zweifach zusammengesetzter Ausdruck (pointing to `b = b * b;`)

Vierfach zusammengesetzter Ausdruck (pointing to the output line)

# Ausdrücke (Expressions)

haben *Typ*, *Wert* und *Effekt* (potenziell).

Beispiel

```
a * a
```

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `a` und `a`
- Effekt: keiner.

Beispiel

```
b = b * b
```

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `b` und `b`
- Effekt: Weise `b` diesen Wert zu.

Typ eines Ausdrucks ist fest, aber Wert und Effekt werden erst durch die *Auswertung* des Ausdrucks bestimmt.

# L-Werte und R-Werte

```
// input
std::cout << "Compute a^8 for a=? ";
int a;
std::cin >> a;

// computation
int b = a * a; // b = a^2
b = b * b; // b = a^4

// output b * b, i.e., a^8
std::cout << a << "^8 = " << b * b << ".\n";
return 0;
```

R-Wert (pointing to the first line)

L-Wert (Ausdruck + Adresse) (pointing to `a;`)

L-Wert (Ausdruck + Adresse) (pointing to `b = a * a;`)

R-Wert (pointing to `b * b` in the computation line)

R-Wert (Ausdruck, der kein L-Wert ist) (pointing to `0;`)

## L-Werte und R-Werte

L-Wert (“**L**inks vom Zuweisungsoperator”)

- Ausdruck mit *Adresse*
- *Wert* ist der Inhalt an der Speicheradresse entsprechend dem Typ des Ausdrucks.
- L-Wert kann seinen Wert ändern (z.B. per Zuweisung).

Beispiel: Variablenname

## L-Werte und R-Werte

R-Wert (“**R**echts vom Zuweisungsoperator”)

- Ausdruck der kein L-Wert ist
- Beispiel: Literal 0
- Jeder L-Wert kann als R-Wert benutzt werden (aber nicht umgekehrt).
  - Ein R-Wert kann seinen Wert *nicht ändern*.

84

85

## Operatoren und Operanden

## Baukasten

```
// input
std::cout << "Compute a^8 for a=? ";
int a;
std::cin >> a;

// computation
int b = a;
b = b * b; // b = a^4

// output
std::cout << a << "^8 = " << b * b << "\n";
return 0;
```

Diagramm zur Analyse des Codes:

- Linker Operand (Ausgabestrom) → `std::cout`
- Ausgabe-Operator → `<<`
- Rechter Operand (String) → `"Compute a^8 for a=? "`
- Rechter Operand (Variablenname) → `a`
- Eingabe-Operator → `>>`
- Linker Operand (Eingabestrom) → `std::cin`
- Zuweisungsoperator → `=`
- Multiplikationsoperator → `*`

86

## Operatoren

Operatoren

- machen aus Ausdrücken (*Operanden*) neue zusammengesetzte Ausdrücke
- spezifizieren für die Operanden und das Ergebnis die Typen, und ob sie L- oder R-Werte sein müssen
- haben eine Stelligkeit

87

## Multiplikationsoperator \*

- erwartet zwei R-Werte vom gleichen Typ als Operanden (Stelligkeit 2)
- "gibt Produkt als R-Wert des gleichen Typs zurück", das heißt formal:
  - Der zusammengesetzte Ausdruck ist ein R-Wert; sein Wert ist das Produkt der Werte der beiden Operanden

Beispiele: `a * a` und `b * b`

88

## Zuweisungsoperator =

- Linker Operand ist L-Wert,
- Rechter Operand ist R-Wert des gleichen Typs.
- Weist linkem Operanden den Wert des rechten Operanden zu und gibt den linken Operanden als L-Wert zurück

Beispiele: `b = b * b` und `a = b`

### Vorsicht, Falle!

Der Operator = entspricht dem Zuweisungsoperator in der Mathematik ( $:=$ ), nicht dem Vergleichsoperator ( $=$ ).

89

## Eingabeoperator >>

- linker Operand ist L-Wert (*Eingabestrom*)
- rechter Operand ist L-Wert
- weist dem rechten Operanden den nächsten Wert aus der Eingabe zu, *entfernt ihn aus der Eingabe* und gibt den Eingabestrom als L-Wert zurück

Beispiel: `std::cin >> a` (meist Tastatureingabe)

- Eingabestrom wird verändert und muss deshalb ein L-Wert sein!

90

## Ausgabeoperator <<

- linker Operand ist L-Wert (*Ausgabestrom*)
- rechter Operand ist R-Wert
- gibt den Wert des rechten Operanden aus, fügt ihn dem Ausgabestrom hinzu und gibt den Ausgabestrom als L-Wert zurück

Beispiel: `std::cout << a` (meist Bildschirmausgabe)

- Ausgabestrom wird verändert und muss deshalb ein L-Wert sein!

91

## Ausgabeoperator <<

Warum Rückgabe des Ausgabestroms?

- erlaubt Bündelung von Ausgaben

```
std::cout << a << "^8 = " << b * b << "\n"
```

ist wie folgt logisch geklammert

```
((((std::cout << a) << "^8 = ") << b * b) << "\n")
```

- `std::cout << a` dient als linker Operand des nächsten << und ist somit ein L-Wert, der kein Variablenname ist.