

# Informatik

Vorlesung am D-ITET/D-MAVT der ETH Zürich

Hermann Lehner, Malte Schwerhoff

FS 2019

## 1. Einführung

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

1

22

### Was ist Informatik?

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

(Wikipedia, nach dem „Duden Informatik“)

### Informatik vs. Computer

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

Mike Fellows, US-Informatiker (1991)

23

24

## Informatik vs. Computer

- 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**.

25

## Informatik $\neq$ EDV-Kenntnisse

EDV-Kenntnisse: *Anwenderwissen* („*Computer Literacy*“)

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

Informatik: *Grundlagenwissen*

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

26

## Zurück in die Gegenwart: Inhalt dieser Vorlesung

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

27

## 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)

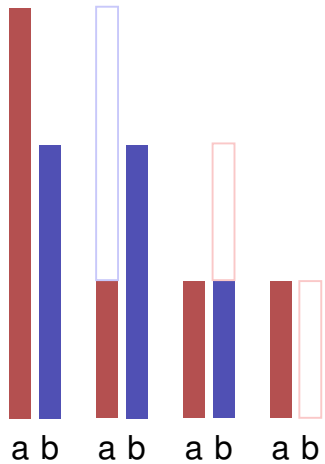


"Dixit algorizmi..." (lateinische Übersetzung)

<http://de.wikipedia.org/wiki/Algorithmus>  
28

# 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:  $a$ .

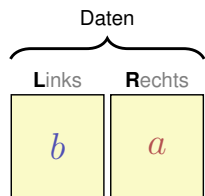
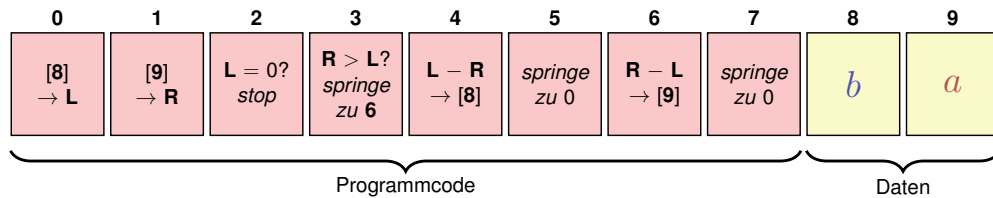
# Algorithmen: 3 Abstraktionsstufen

- 1. Kernidee** (abstrakt): Die Essenz eines Algorithmus' („Heureka-Moment“)
- 2. Pseudocode** (semi-detailliert): Für Menschen gemacht (Bildung, Korrektheit- und Effizienzdiskussionen, Beweise)
- 3. Implementierung** (sehr detailliert): Für Mensch & Computer gemacht (les- & ausführbar, bestimmte Programmiersprache, verschiedene Implementierungen möglich)

Euklid: Kernidee und Pseudocode gesehen, Implementierung noch nicht

# Euklid in der Box

Speicher



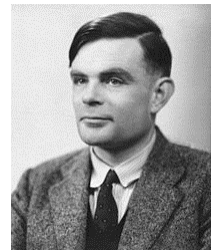
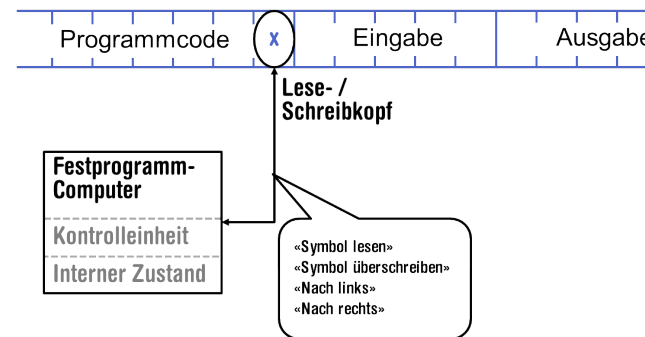
Register

Solange  $b \neq 0$   
 Wenn  $a > b$  dann  
 $a \leftarrow a - b$   
 Sonst:  
 $b \leftarrow b - a$   
 Ergebnis:  $a$ .

# Computer – Konzept

Eine geniale Idee: Universelle Turingmaschine (Alan Turing, 1936)

Folge von Symbolen auf Ein- und Ausgabeband

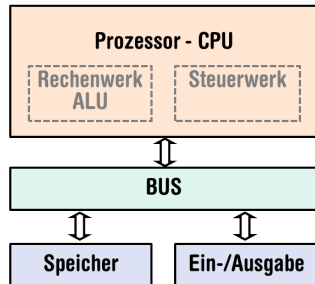


Alan Turing

## Computer – Umsetzung

- Z1 – Konrad Zuse (1938)
- ENIAC – John Von Neumann (1945)

### Von Neumann Architektur



Konrad Zuse



John von Neumann

<http://www.hs.uni-hamburg.de/DE/UNT/HH/biogr/zuse.htm>  
[http://commons.wikimedia.org/wiki/File:John\\_von\\_Neumann.jpg](http://commons.wikimedia.org/wiki/File:John_von_Neumann.jpg)

33

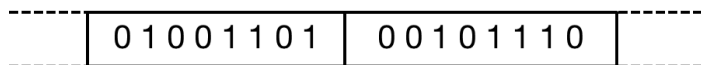
## Computer

### 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).



Adresse : 17      Adresse : 18

## 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)

35

34

36

## 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)

37

## 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>1</sup>

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

## 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 ...
- ...

*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*

(Lino Guzzella ist übrigens nicht Informatiker, sondern Maschineningenieur und Prof. für Thermotronik ☺)

## 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 (und ist nützlich)!

41

## Programmiersprachen

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

42

## Höhere Programmiersprachen

darstellbar als Programmtext, der

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

43

## Programmiersprachen – Einordnung

Unterscheidung in

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

44

## Warum C++?

Andere populäre höhere Programmiersprachen: Java, C#, Python, Javascript, Swift, Kotlin, Go, ... . . .

Allgemeiner Konsens

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

45

## 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

46

## 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

47

## Syntax und Semantik

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

48

## Deutsch vs. C++

### Deutsch

*Alleen sind nicht gefährlich, Rasen ist gefährlich!*  
(Wikipedia: Mehrdeutigkeit)

### C++

```
// computation
int b = a * a; // b = a2
b = b * b;    // b = a4
```

49

## 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 galoppiert 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]

50

## Syntax und Semantik von C++

### Syntax:

- Wann ist ein Text ein C++-Programm?
- D.h. ist es *grammatikalisch* korrekt?
- → Kann vom Computer überprüft werden

### Semantik:

- Was *bedeutet* ein Programm?
- Welchen Algorithmus *implementiert* ein Programm?
- → Braucht menschliches Verständnis

51

## Syntax und Semantik von C++

Der ISO/IEC Standard 14822 (1998, 2011, 2014, ...)

- ist das „Gesetz“ von C++
- legt Grammatik und Bedeutung von C++-Programmen fest
- wird seit 2011 regelmässig durch Neuerungen für *fortgeschrittenes* Programmieren erweitert

52



## Was braucht es zum Programmieren?

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

53

## 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

54

## 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;
}
```

55

## 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)

56

## „Beiwerk“: Kommentare

```
// 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;
    // 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;
}
```

Kommentare

57

## 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)

58

## 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!

59

## „Beiwerk“: Include und Main-Funktion

```
// 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;
    // 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;
}
```

Include-Direktive  
Funktionsdeklaration der main-Funktion

60

## Include-Direktiven

C++ besteht aus

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

```
#include <iostream>
```

- macht Ein/Ausgabe verfügbar

## 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.

61

62

## 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

## Anweisungen

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

63

64

## 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;`

## Rückgabeeanweisungen

- treten nur in Funktionen auf und sind von der Form  
`return expr;`  
wobei *expr* ein Ausdruck ist
- spezifizieren Rückgabewert der Funktion

Beispiel: `return 0;`

65

66

## Anweisungen – Effekte

```
int main() {  
  // input  
  std::cout << "Compute a^8 for a =? ";  
  int a;  
  std::cin >> a;  
  // computation  
  int b = a * a;  
  b = b * b;  
  // 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*

67

## 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

68

## Anweisungen – Variablendefinitionen

```
int main() {
    // input
    std::cout << "Compute a^8 for a=? ";
    int a; // Deklarationsanweisungen
    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;
}
```

Typ-namen

## Deklarationsanweisungen

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

Beispiel: `int a;`

- können Variablen auch initialisieren

Beispiel: `int b = a * a;`

69

70

## 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)

## Fundamentaltypen

C++ enthält fundamentale Typen für

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

71

72

## 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  $1.2 \cdot 10^5$ .

73

## 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

74

## 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 (→ Informatik 1).

75

## 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)

76

## 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

## Ausdrücke

## Baukasten

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

// computation
int b = a * a; // b = a^2
b = b * b; // Zweifach zusammengesetzter Ausdruck

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

Zusammengesetzter Ausdruck

Zweifach zusammengesetzter Ausdruck

Vierfach zusammengesetzter Ausdruck

77

78

## 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)
```

79

## 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.

80

## 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

L-Wert (Ausdruck + Adresse)

L-Wert (Ausdruck + Adresse)

R-Wert

R-Wert (Ausdruck, der kein L-Wert ist)

81

## L-Werte und R-Werte

L-Wert (“Links 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

82

## L-Werte und R-Werte

R-Wert (“Rechts 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*.

83

## Operatoren und Operanden

## Baukasten

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

Linker Operand (Ausgabestrom)

Ausgabe-Operator

Rechter Operand (String)

Rechter Operand (Variablenname)

Eingabe-Operator

Linker Operand (Eingabestrom)

Zuweisungsoperator

Multiplikationsoperator

84



## 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

## 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  $b * b$

85

86

## 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 ( $=$ ).

## 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!

87

88

## 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!

## 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.