

# Informatik

Vorlesung für Rechnergestützte Wissenschaften  
am D-MATH der ETH Zürich

Felix Friedrich, Malte Schwerhoff

HS 2018

# 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?

# Was ist Informatik?

- Die Wissenschaft der **systematischen Verarbeitung von Informationen**, . . .

# 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)

# Informatik vs. Computer

- Die Informatik beschäftigt sich heute auch mit dem Entwurf von schnellen Computern und Netzwerken. . .

# 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.**

# Informatik $\neq$ EDV-Kenntnisse

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

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

# Informatik $\neq$ EDV-Kenntnisse

Informatik: *Grundlagenwissen*

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

# Inhalt dieser Vorlesung

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

# Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Handlungsanweisung zur schrittweisen Lösung eines Problems

# Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Handlungsanweisung zur schrittweisen Lösung eines Problems
- Ausführung erfordert keine Intelligenz, nur Genauigkeit (sogar Computer können es)

# 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)

# 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:  $a$ .



# 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  $b \neq 0$

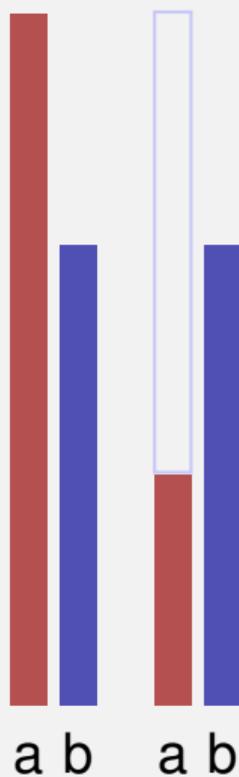
Wenn  $a > b$  dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .



# 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  $b \neq 0$

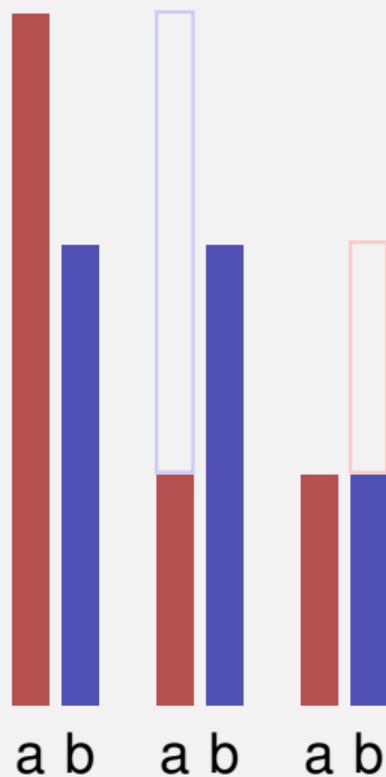
Wenn  $a > b$  dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .



# 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  $b \neq 0$

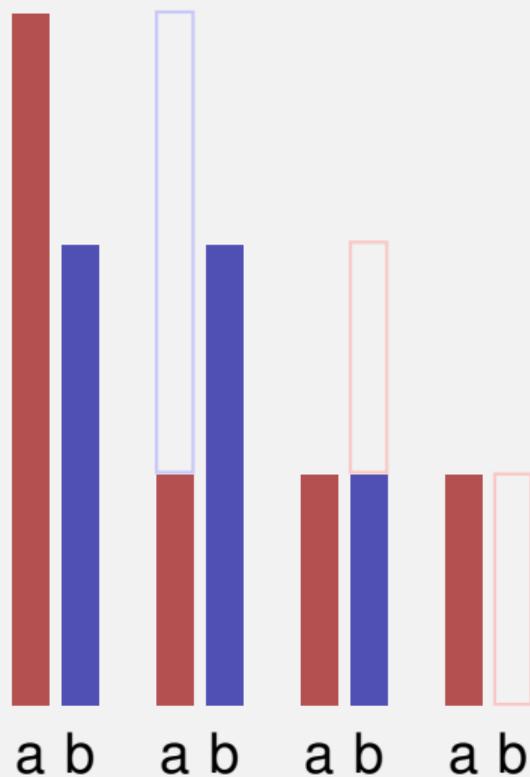
Wenn  $a > b$  dann

$$a \leftarrow a - b$$

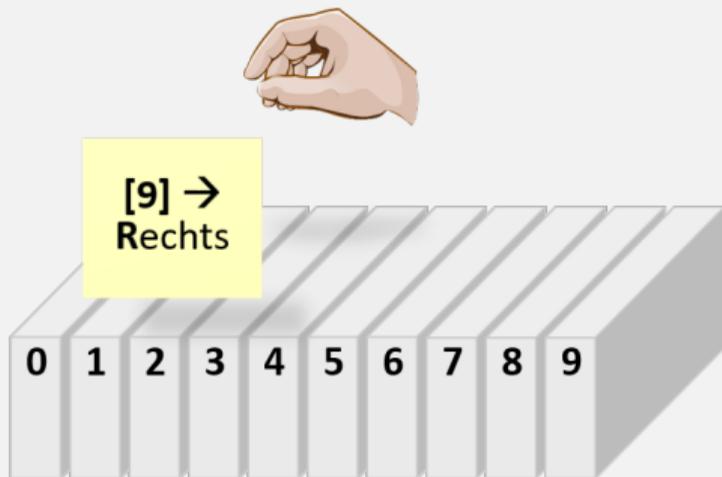
Sonst:

$$b \leftarrow b - a$$

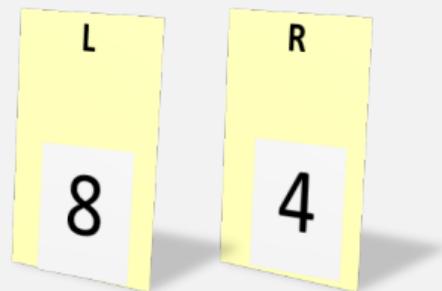
Ergebnis:  $a$ .



# Live Demo: Turing Maschine



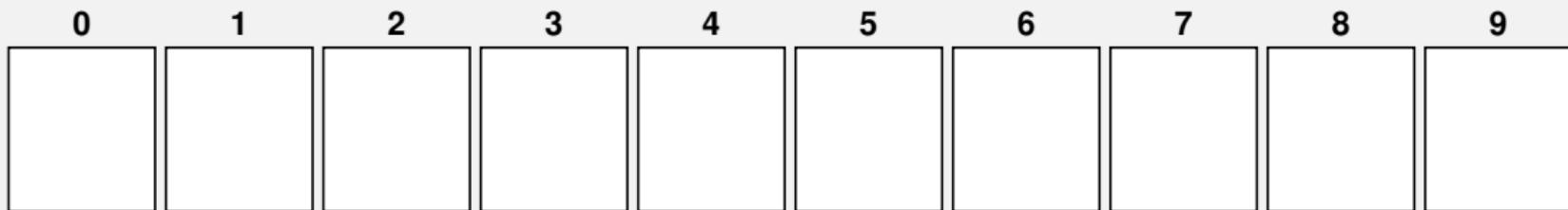
Speicher



Register

# Euklid in der Box

*Speicher*



**L**inks

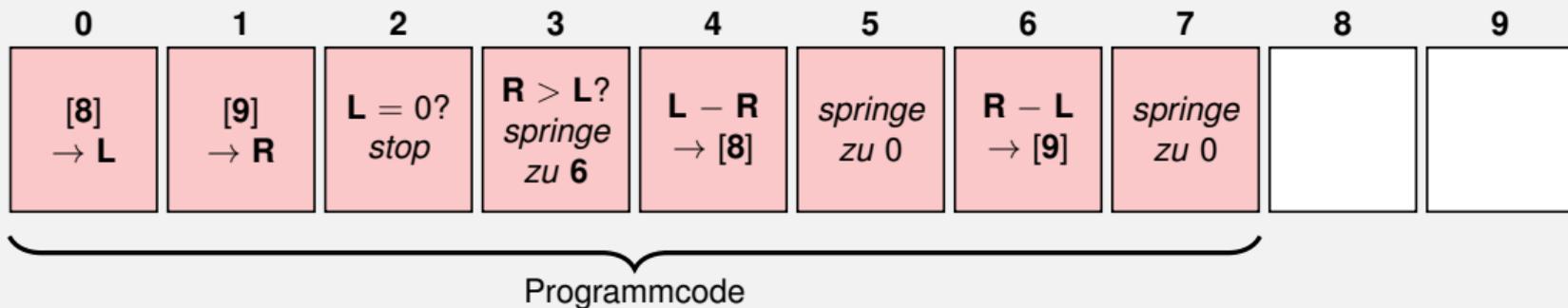
**R**echts



*Register*

# Euklid in der Box

*Speicher*



Links

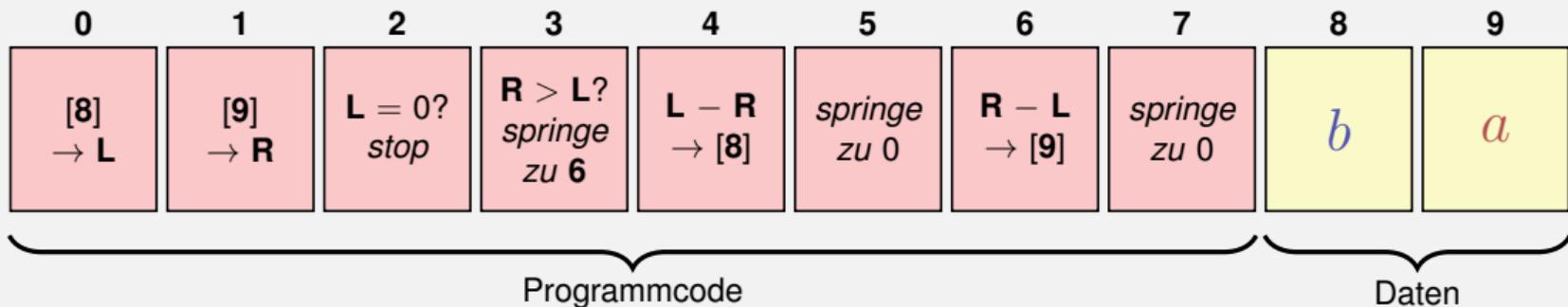
Rechts



*Register*

# Euklid in der Box

Speicher



Links

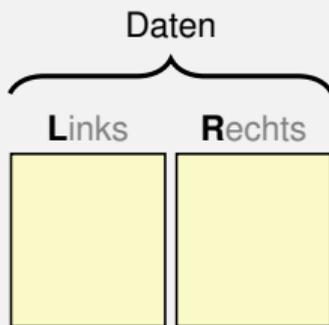
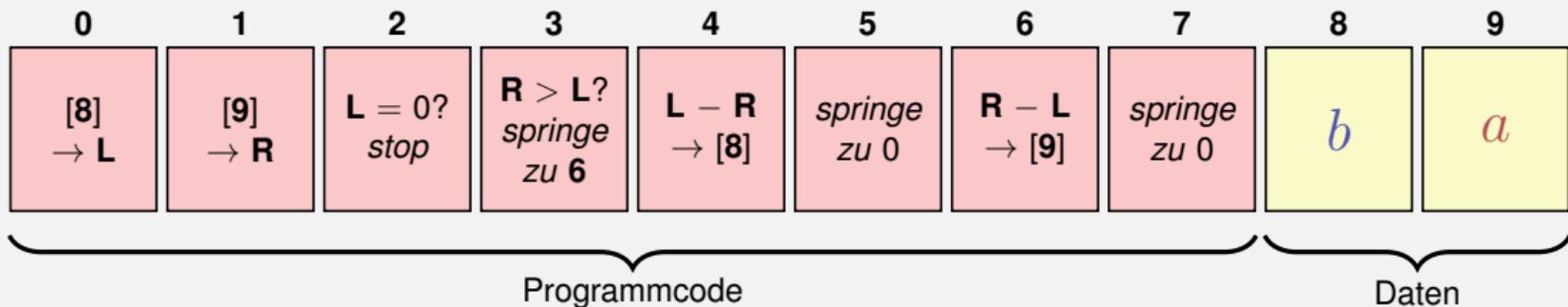
Rechts



Register

# Euklid in der Box

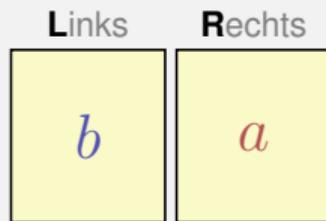
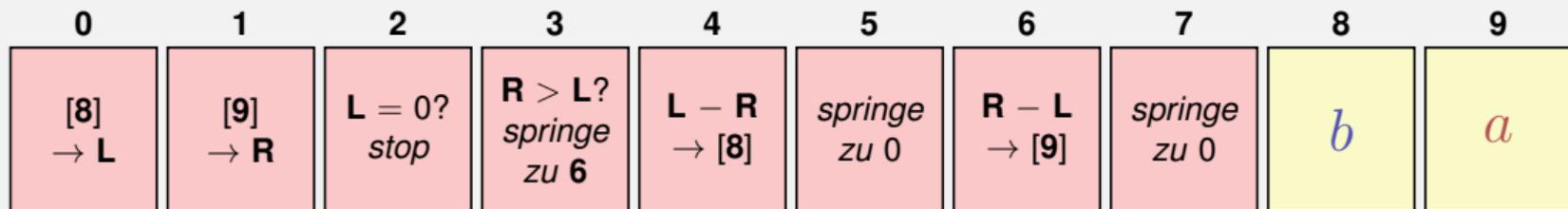
Speicher



Register

# Euklid in der Box

## Speicher



## Register

Solange  $b \neq 0$

Wenn  $a > b$  dann

$$a \leftarrow a - b$$

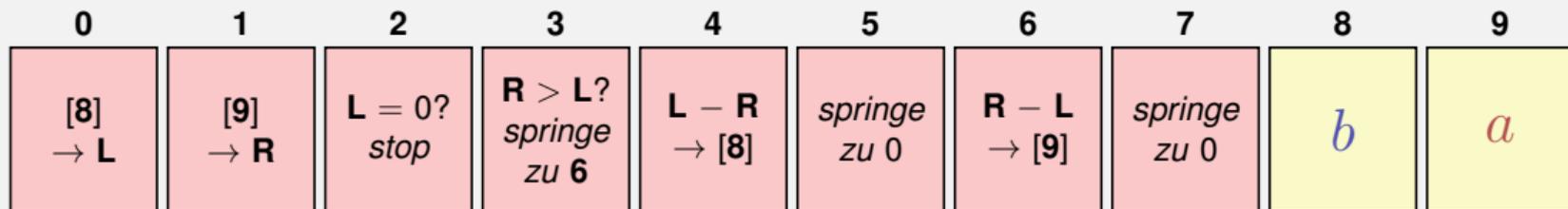
Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .

# Euklid in der Box

Speicher



Solange  $b \neq 0$

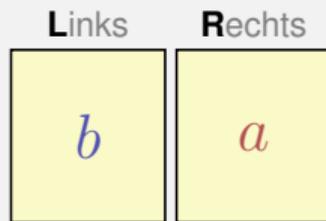
Wenn  $a > b$  dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

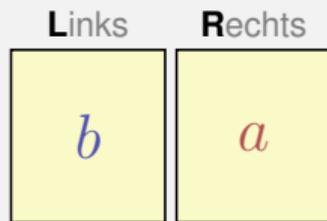
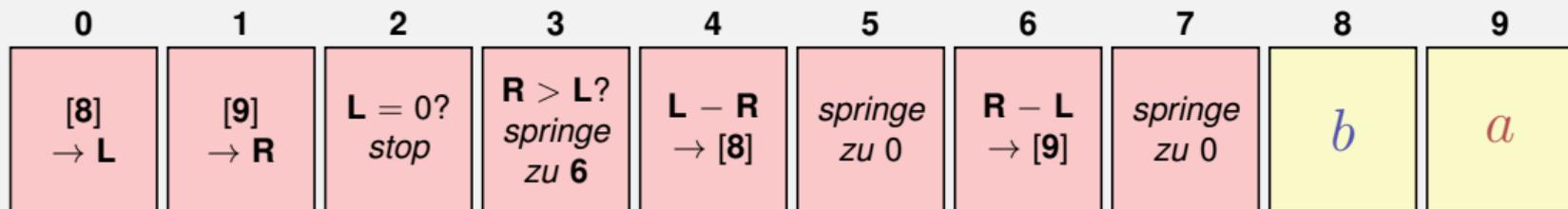
Ergebnis:  $a$ .



Register

# Euklid in der Box

Speicher



Register

Solange  $b \neq 0$

Wenn  $a > b$  dann

$$a \leftarrow a - b$$

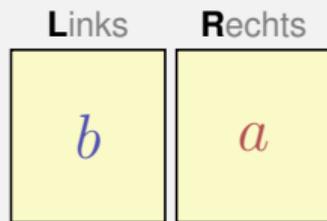
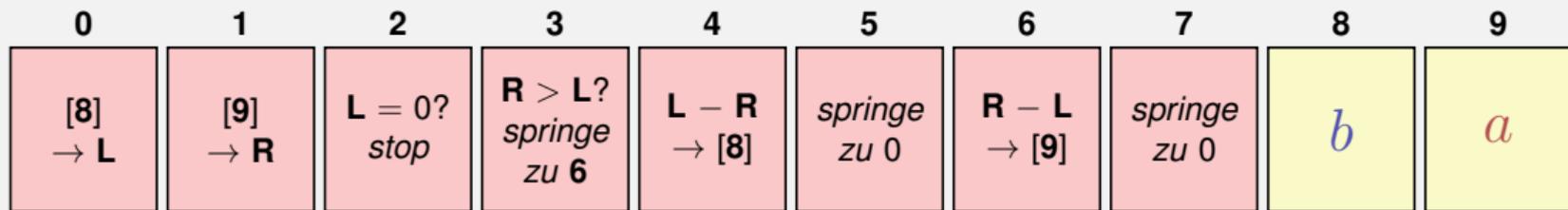
Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .

# Euklid in der Box

Speicher



Register

Solange  $b \neq 0$

Wenn  $a > b$  dann

$$a \leftarrow a - b$$

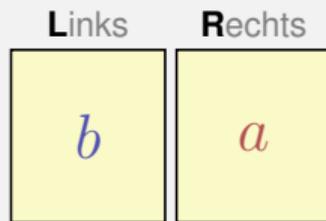
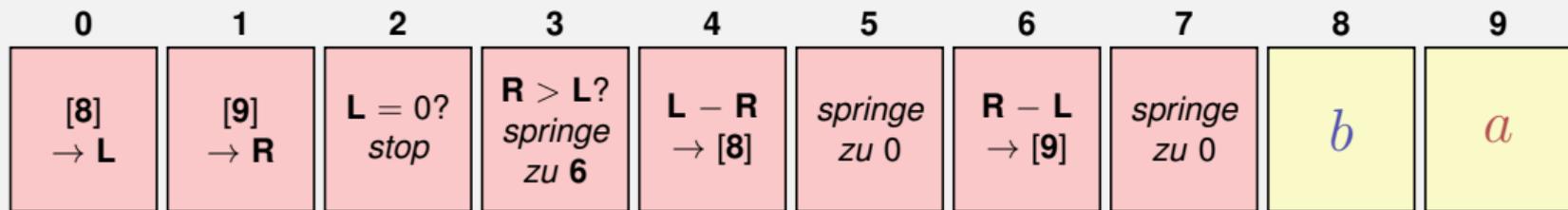
Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .

# Euklid in der Box

Speicher



Register

Solange  $b \neq 0$

Wenn  $a > b$  dann

$$a \leftarrow a - b$$

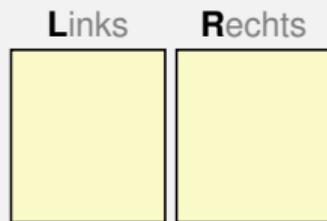
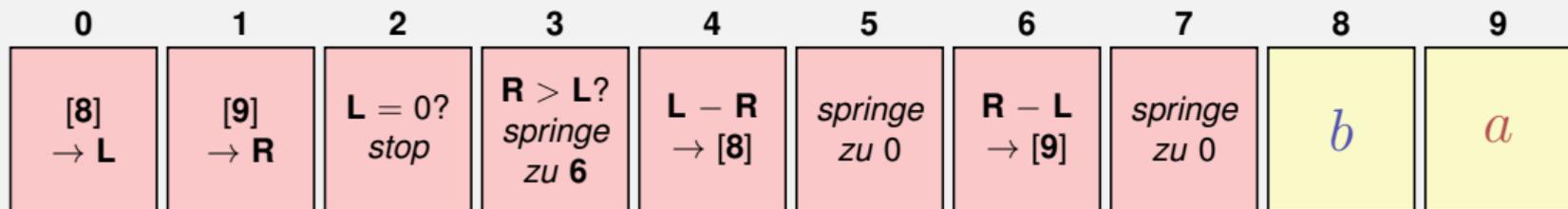
Sonst:

$$b \leftarrow b - a$$

Ergebnis:  $a$ .

# Euklid in der Box

## Speicher



## Register

Solange  $b \neq 0$

Wenn  $a > b$  dann

$$a \leftarrow a - b$$

Sonst:

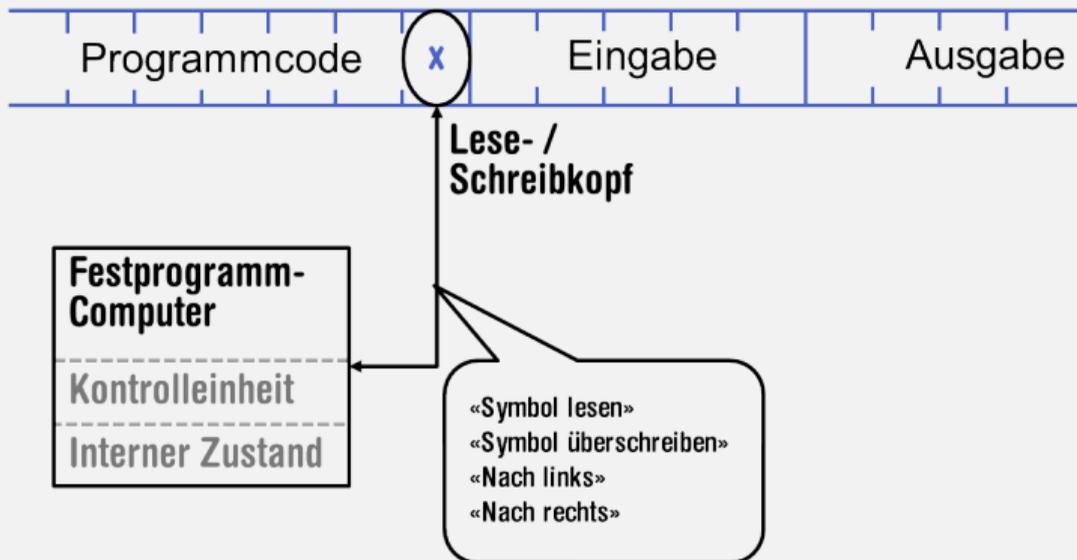
$$b \leftarrow b - a$$

Ergebnis:  $a$ .

# Computer – Konzept

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

Folge von Symbolen auf Ein- und Ausgabeband

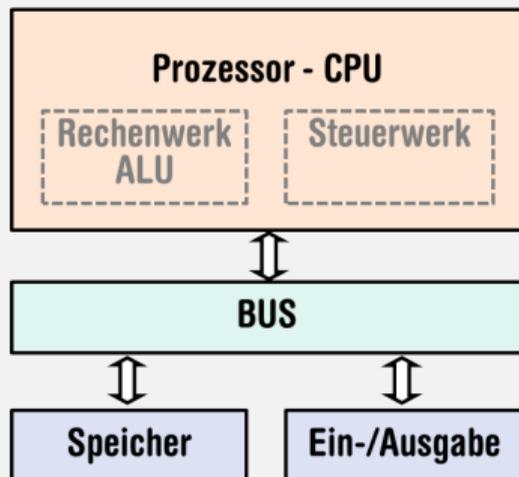


Alan Turing

# Computer – Umsetzung

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

## Von Neumann Architektur



Konrad Zuse



John von Neumann

# 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).

# Speicher für Daten *und* Programm

- Jede Speicherzelle hat eine Adresse.
- Random Access: Zugriffszeit auf Speicherzelle (nahezu) unabhängig von ihrer Adresse.



# Rechengeschwindigkeit

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

---

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

# Rechengeschwindigkeit

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



30 m

arbeitet ein heutiger Desktop-PC mehr als 100

---

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

# 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

# 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

# Warum Programmieren?

- Da hätte ich ja gleich Informatik studieren können ...

# Warum Programmieren?

- Da hätte ich ja gleich Informatik studieren können ...
- Es gibt doch schon für alles Programme ...

# Warum Programmieren?

- Da hätte ich ja gleich Informatik studieren können ...
- Es gibt doch schon für alles Programme ...
- Programmieren interessiert mich nicht ...

# 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 ...

# 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)

# 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.

# 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)!

# 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.

# Höhere Programmiersprachen

darstellbar als Programmtext, der

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

# Warum C++?

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

# Warum C++?

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

Allgemeiner Konsens

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

# Warum C++?

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

# 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 = a2  
b = b * b;    // b = a4
```

# Syntax und Semantik

- Programme müssen, wie unsere Sprache, nach gewissen Regeln geformt werden.
  - **Syntax**: Zusammenfügungsregeln für elementare Zeichen (Buchstaben).
  - **Semantik**: Interpretationsregeln für zusammengefügte Zeichen.

# Syntax und Semantik

- Entsprechende Regeln für ein Computerprogramm sind einfacher, aber auch strenger, denn Computer sind vergleichsweise dumm.

# Syntax und Semantik von C++

## *Syntax*

- Wann ist ein Text ein C++ Programm?
- D.h. ist es *grammatikalisch* korrekt?

## *Semantik*

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

# Was braucht es zum Programmieren?

- **Editor:** Programm zum Ändern, Erfassen und Speichern von C++-Programmtext
- **Compiler:** Programm zum Übersetzen des Programmtexts in Maschinsprache

# Was braucht es zum Programmieren?

- **Computer:** Gerät zum Ausführen von Programmen in Maschinensprache
- **Betriebssystem:** Programm zur Organisation all dieser Abläufe (Dateiverwaltung, Editor-, Compiler- und Programmaufruf)

# Das erste C++ Programm

```
// 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;
}
```

```
// 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;  Mache etwas (lies a ein)!
    // 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;
}
```

```
// 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 ←———— 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;
}
```

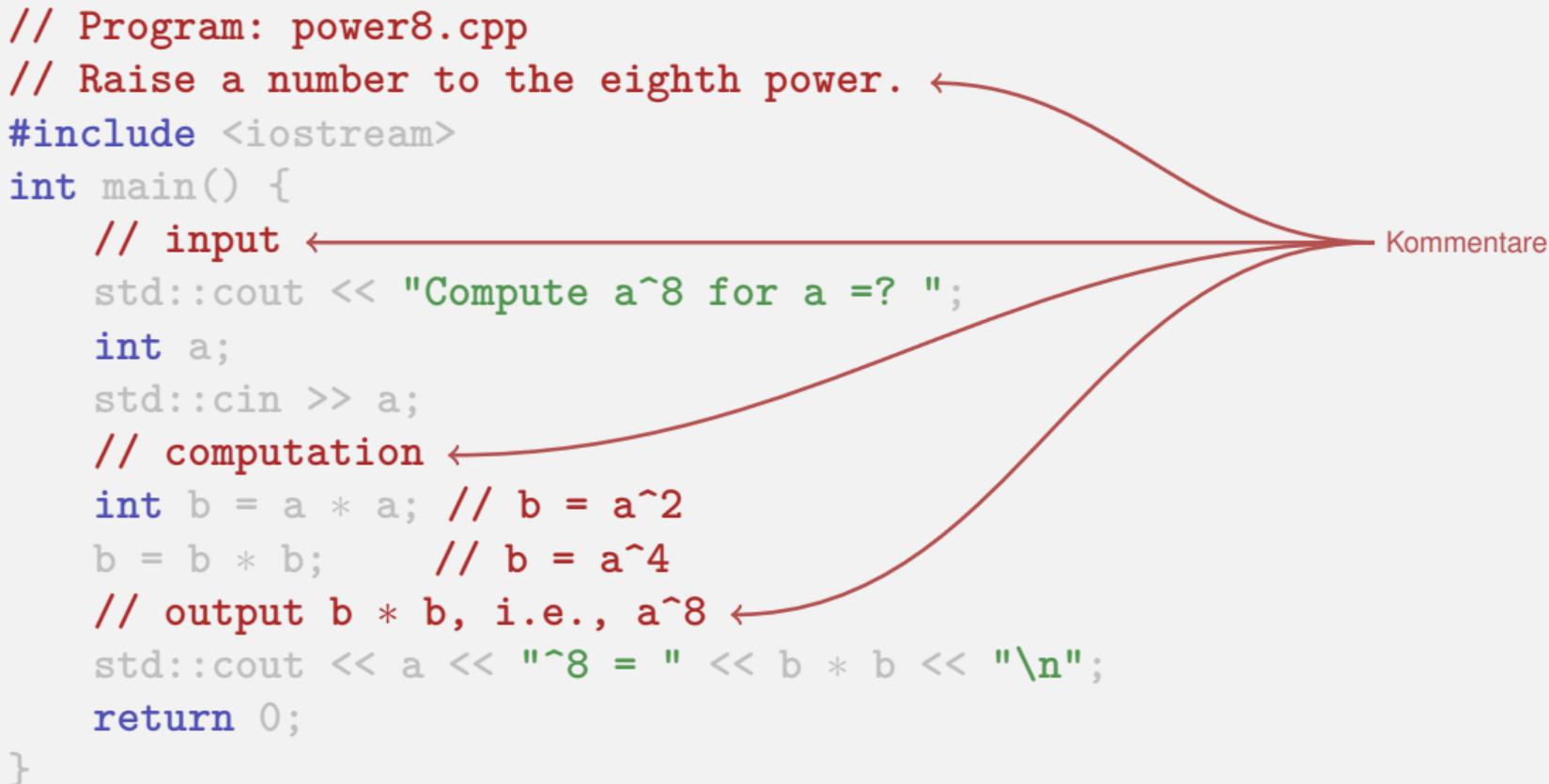
# “Beiwerk”: Kommentare

```
// 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;
}
```

# “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



## 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;}
```

---

## 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!**

# „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;
}
```

# „Beiwerk“: Include und Main-Funktion

```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream> ← Include-Direktive
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;
}
```

# „Beiwerk“: Include und Main-Funktion

```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
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;
}
```

# 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;  
}
```

# 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

# 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; ← Rückgabeeweisung  
}
```

# 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*

# 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

# 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  $1.2 \cdot 10^5$ .

# Variablen

- repräsentieren (wechselnde) Werte
- haben
  - *Name*
  - *Typ*
  - *Wert*
  - *Adresse*

# Variablen

- repräsentieren (wechselnde) Werte
- haben
  - *Name*
  - *Typ*
  - *Wert*
  - *Adresse*

## Beispiel

`int a;` definiert Variable mit

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

# Ausdrücke: Berechne einen Wert!

- repräsentieren *Berechnungen*

# Ausdrücke: Berechne einen Wert!

- repräsentieren *Berechnungen*
- sind entweder **primär** (b)

# Ausdrücke: Berechne einen Wert!

- repräsentieren *Berechnungen*
- sind entweder **primär** ( $b$ )
- oder **zusammengesetzt** ( $b*b$ )...

# Ausdrücke: Berechne einen Wert!

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

# 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

# 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

```
// 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;
```

# Ausdrücke

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

— Variablenname, primärer Ausdruck (+ Name und Adresse)

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

— Variablenname, primärer Ausdruck (+ Name und Adresse)

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

```
return 0;
```

— Literal, primärer Ausdruck

Zusammengesetzter Ausdruck

```
// 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;
```

```
// 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

# 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;
```

# 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;
```

L-Wert (Ausdruck + Adresse)

L-Wert (Ausdruck + Adresse)

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

# 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;
```

The image illustrates the evaluation of expressions in C++ code, highlighting the use of Right-Value (R-Value) references. Red boxes and arrows indicate the following:

- The string literal `"Compute a^8 for a =? "` is boxed and labeled as an R-Value.
- The expression `b * b` in the assignment `b = b * b;` is boxed and labeled as an R-Value.
- The expression `b * b` in the output statement `std::cout << a << "^8 = " << b * b << ".\n";` is boxed and labeled as an R-Value.

# 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-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

# 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).

# 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).

Jedes e-Bike kann als normales Fahrrad benutzt werden, aber nicht umgekehrt.

# L-Werte und R-Werte

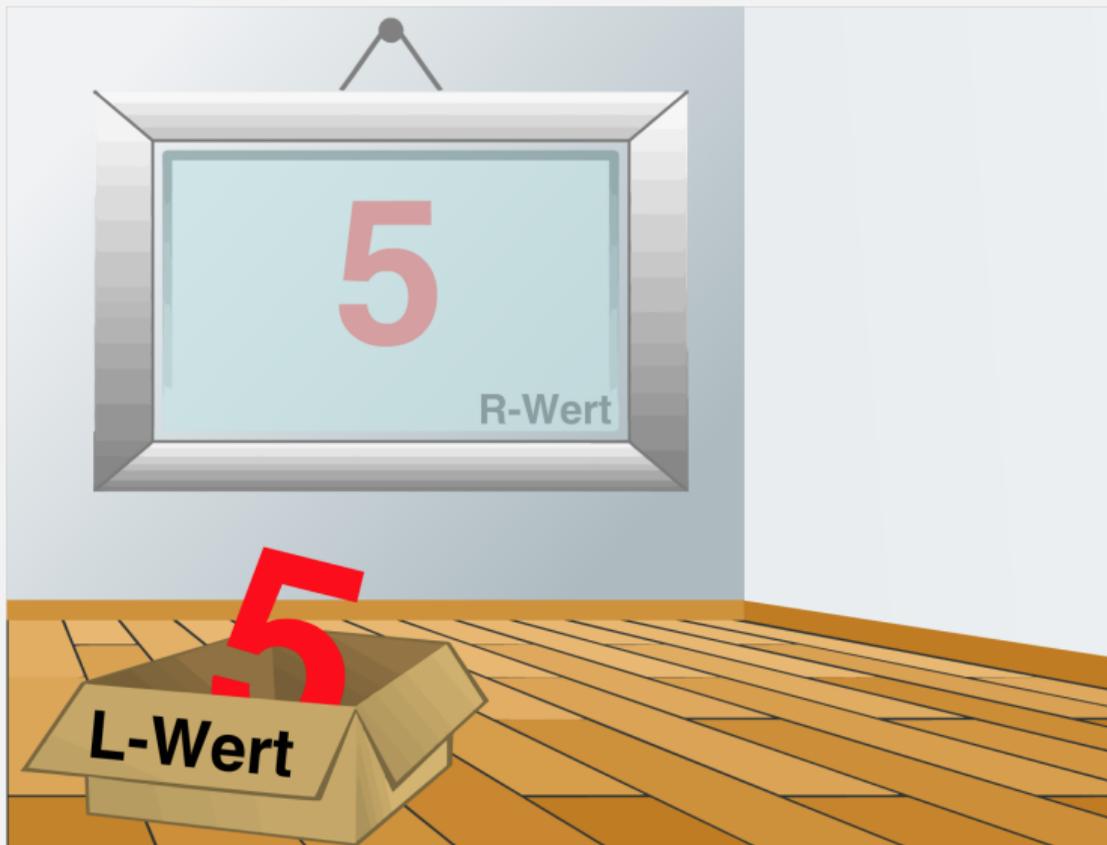
R-Wert (“**R**echts vom Zuweisungsoperator”)

- 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*.

# 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;
```

```
// 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;
```

Linker Operand (Ausgabestrom)

Ausgabe-Operator

Rechter Operand (String)

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

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

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

Rechter Operand (Variablenname)

Eingabe-Operator

Linker Operand (Eingabetrom)

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

**Multiplikationsoperator**