ETH zürich

Felix Friedrich, Malte Schwerhoff

Informatik

Vorlesung am D-MATH/D-PHYS der ETH Zürich

Herbst 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

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?

Zurück in die Gegenwart: 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)

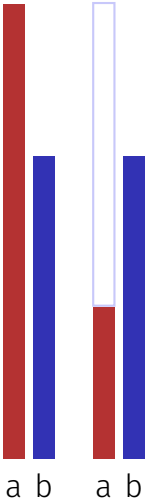
Der älteste nichttriviale Algorithmus

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

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

Der älteste nichttriviale Algorithmus

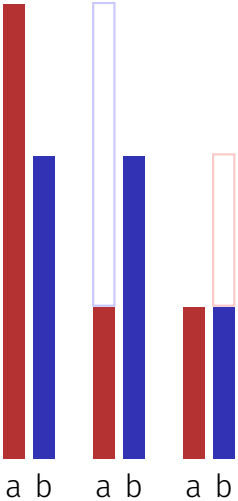
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



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

Der älteste nichttriviale Algorithmus

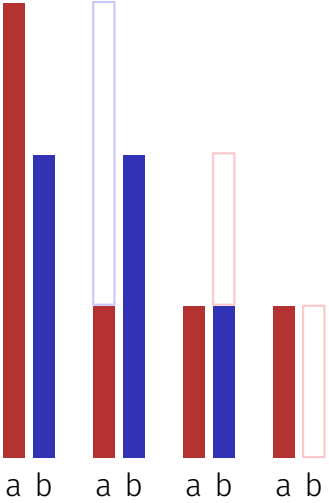
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



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

Der älteste nichttriviale Algorithmus

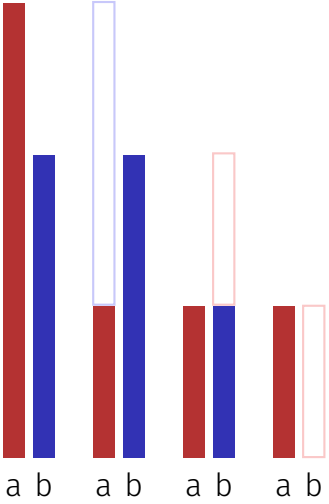
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



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

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")

Algorithmen: 3 Abstraktionsstufen

1. **Kernidee** (abstrakt)**:**
   Die Essenz eines Algorithmus' („Heureka-Moment")
2. **Pseudocode** (semi-detailliert)**:**
   Für Menschen gemacht (Bildung, Korrektheit- und Effizienzdiskussionen, Beweise)

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)

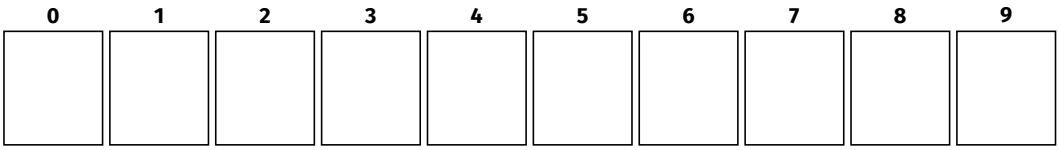
Algorithmen: 3 Abstraktionsstufen

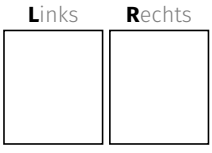
1. **Kernidee** (abstrakt)**:**
   Die Essenz eines Algorithmus' („Heureka-Moment")
2. **Pseudocode** (semi-detailliert)**:**
   Für Menschen gemacht (Bildung, Korrektheit- und Effizienzdiskussionen, Beweise)
3. **Implementierung** (sehr detailliert)**:**
   Für Mensch & Computer gemacht (les- & ausführbar,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

Euklid: Kernidee und Pseudocode gesehen, Implementierung noch nicht

Euklid in der Box

Speicher

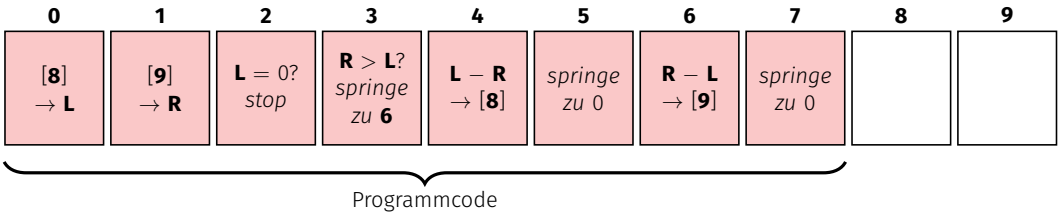


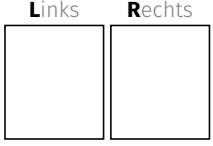


Register

Euklid in der Box

Speicher

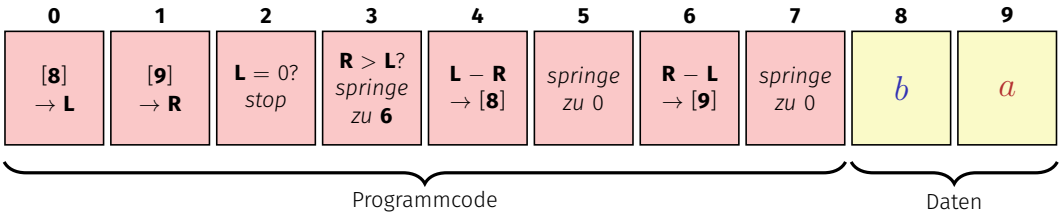


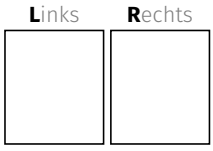


Register

Euklid in der Box

Speicher

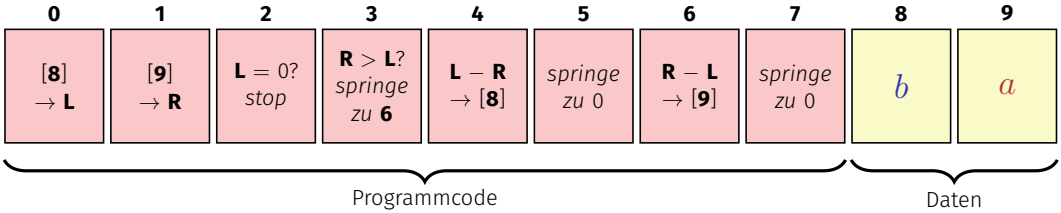


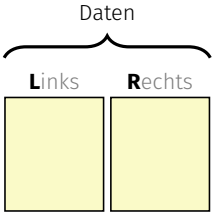


Register

Euklid in der Box

Speicher





Register

Euklid in der Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a

Links **R**echts

b	a

Register

Solange $b \neq 0$

Wenn $a > b$ dann

$a \leftarrow a - b$

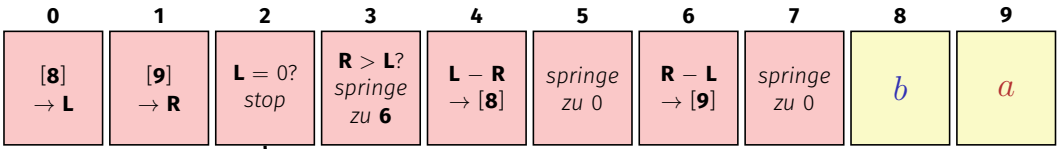
Sonst:

$b \leftarrow b - a$

Ergebnis: a.

Euklid in der Box

Speicher



Solange $b \neq 0$

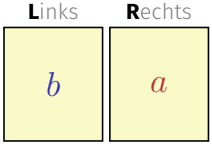
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

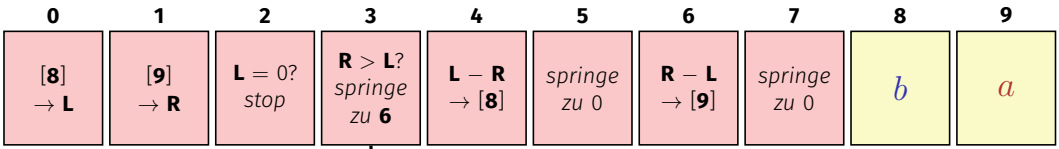
Ergebnis: a.



Register

Euklid in der Box

Speicher



Solange $b \neq 0$

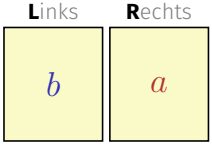
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

Ergebnis: a.

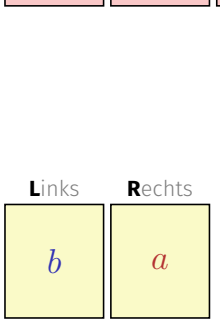


Register

Euklid in der Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a

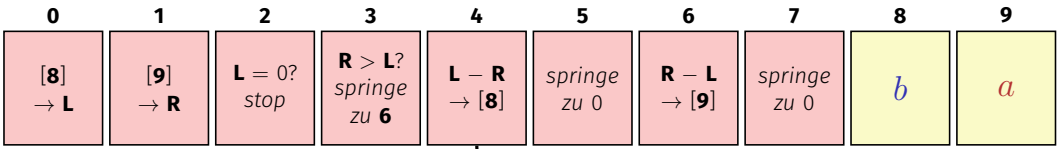


Register

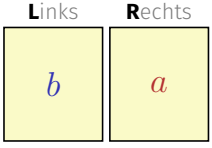
Solange $b \neq 0$
Wenn $a > b$ dann
$a \leftarrow a - b$
Sonst:
$b \leftarrow b - a$
Ergebnis: a.

Euklid in der Box

Speicher



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

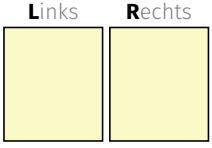


Register

Euklid in der Box

Speicher

0	1	2	3	4	5	6	7	8	9
[**8**] → **L**	[**9**] → **R**	**L** = 0? *stop*	**R** > **L**? *springe zu* **6**	**L** − **R** → [**8**]	*springe zu* 0	**R** − **L** → [**9**]	*springe zu* 0	b	a



Register

Solange $b \neq 0$

Wenn $a > b$ dann

$a \leftarrow a - b$

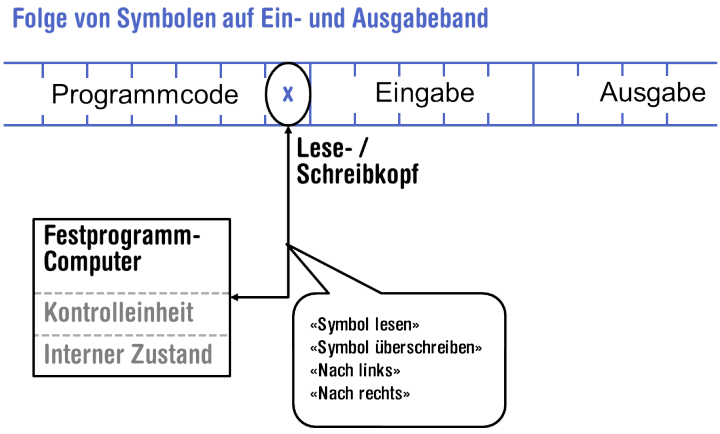
Sonst:

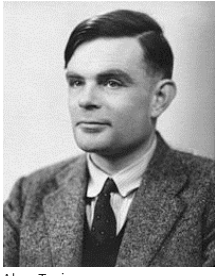
$b \leftarrow b - a$

Ergebnis: a.

Computer – Konzept

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

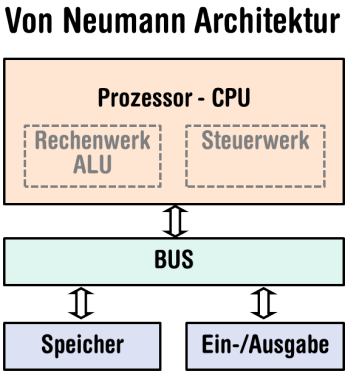




Alan Turing

Computer – Umsetzung

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




Konrad Zuse

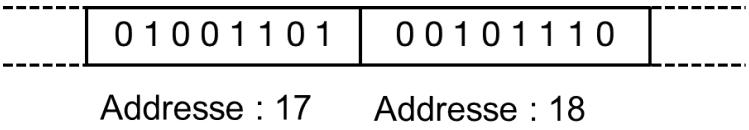

John von Neumann

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.



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

Rechengeschwindigkeit

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

[1] Uniprozessor Computer bei 1GHz

Rechengeschwindigkeit

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

arbeitet ein heutiger Desktop-PC mehr als 100

[1] Uniprozessor Computer bei 1GHz

Rechengeschwindigkeit

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

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

arbeitet ein heutiger Desktop-PC mehr als 100 Millionen Instruktionen ab.[1]

[1] Uniprozessor Computer bei 1GHz

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 2015-2018, 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

  $\rightarrow$ Abstraktion!

Warum C++?

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

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.

Warum C++?

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

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

- 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.

Deutsch vs. C++

Deutsch

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

C++

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

Syntax und Semantik von C++

Syntax:

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

Semantik:

- Was *bedeutet* ein Programm?
- Welchen Algorithmus *implementiert* ein Programm?
- $\rightarrow$ Braucht menschliches Verständnis

Was braucht es zum Programmieren?

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

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)

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;
}
```

Wichtigste Bestandteile... Anweisungen

```
// 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;
}
```

Mache etwas (lies a ein)!

```
// 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;
}
```

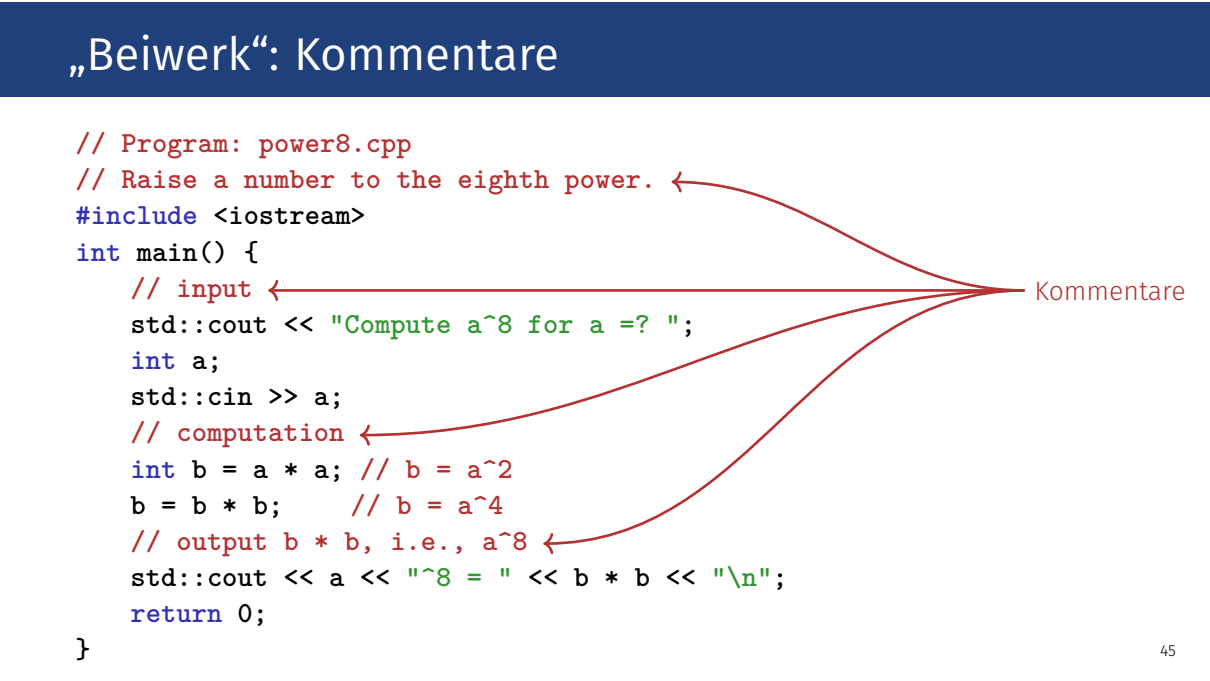
Berechne einen Wert (a^2)!

„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 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;}
```

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!

„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>
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

„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;
}
```

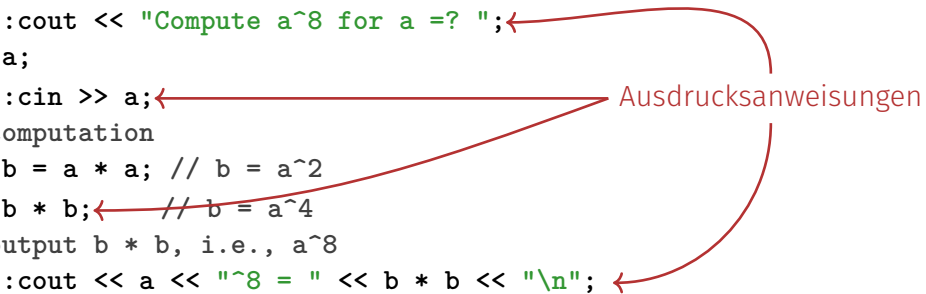
Funktionsdeklaration der main-Funktion

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;
}
```



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;
}
```

Rückgabeanweisung

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 $a \cdot a$ in b

Effekt: Speichern des berechneten Wertes von $b \cdot b$ in b

Effekt: Rückgabe des Wertes 0

Effekt: Ausgabe des Wertes von a und des berechneten Wertes von $b \cdot b$

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;
}
```

Deklarationsanweisungen

Typ-namen

Variablen

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

Variablen

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

`int a;` definiert Variable mit

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

Ausdrücke: Berechne einen Wert!

Ausdrücke

- repräsentieren *Berechnungen*

Ausdrücke: Berechne einen Wert!

Ausdrücke

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

Ausdrücke: Berechne einen Wert!

Ausdrücke

- repräsentieren *Berechnungen*
- sind entweder **primär** (`b`)
- oder **zusammengesetzt** (`b*b`)…

Ausdrücke: Berechne einen Wert!

Ausdrücke

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

Ausdrücke: Berechne einen Wert!

Ausdrücke

- 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!

Ausdrücke

- 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;

// 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;
```

Variablenname, primärer Ausdruck (+ Name und Adresse)

Variablenname, primärer Ausdruck (+ Name und Adresse)

Literal, primärer Ausdruck

Ausdrücke

Baukasten

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

return 0;
```

Zusammengesetzer Ausdruck

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

Zweifach zusammengesetzter Ausdruck

Vierfach zusammengesetzter Ausdruck

Literale

- repräsentieren konstante Werte
- haben festen **Typ** und **Wert**
- sind „syntaktische Werte“

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

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

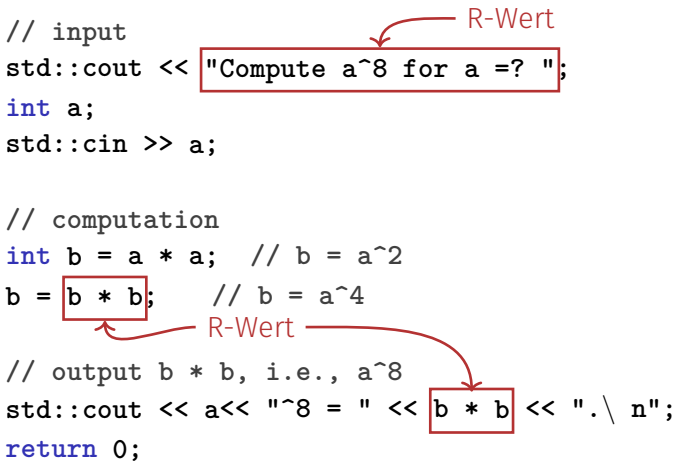
```
// input
std::cout << "Compute a^8 for a =? ";
int a;
std::cin >> a;

// 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



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
- Jeder L-Wert kann als R-Wert benutzt werden (aber nicht umgekehrt).

Beispiel: Literal `0`

L-Werte und R-Werte

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

- Ausdruck der kein L-Wert ist
- Jeder L-Wert kann als R-Wert benutzt werden (aber nicht umgekehrt). Jedes e-Bike kann als normales Fahrrad benutzt werden, aber nicht umgekehrt.

Beispiel: Literal `0`

L-Werte und R-Werte

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

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

Beispiel: Literal `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;
```

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

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 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

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

Zuweisungsoperator

Multiplikationsoperator