

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 \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. . .

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 \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 $b \neq 0$

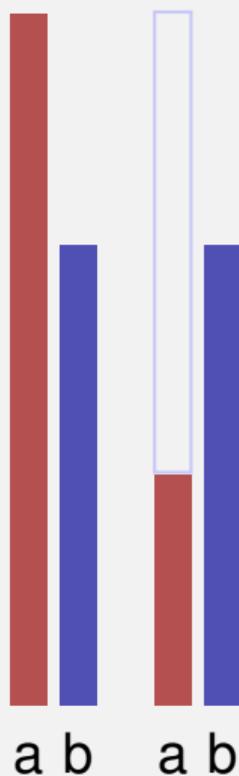
Wenn $a > b$ dann

$$a \leftarrow a - b$$

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 $b \neq 0$

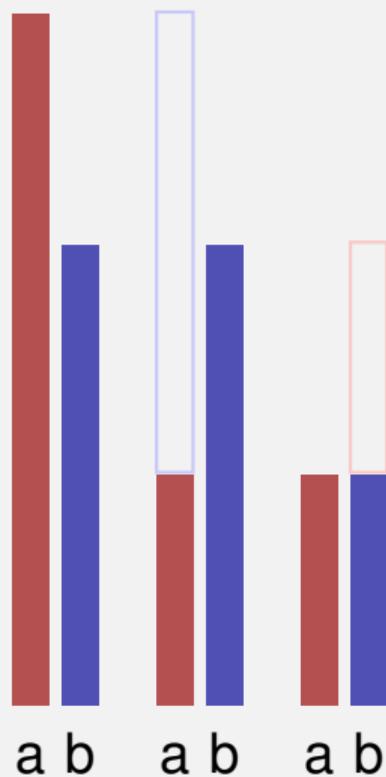
Wenn $a > b$ dann

$$a \leftarrow a - b$$

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 $b \neq 0$

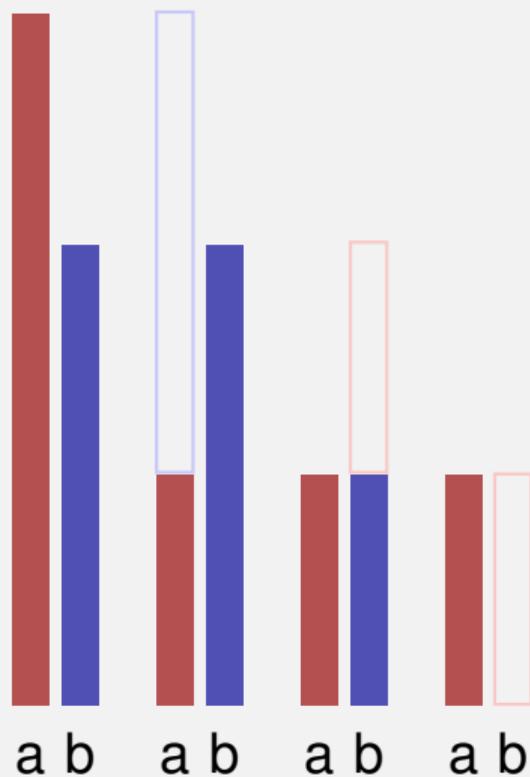
Wenn $a > b$ dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

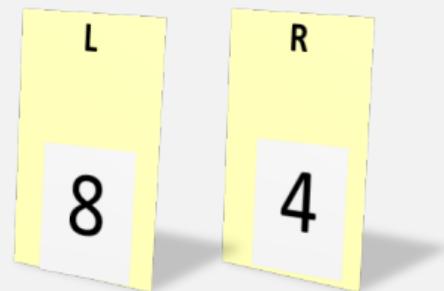
Ergebnis: a .



Live Demo: Turing Maschine



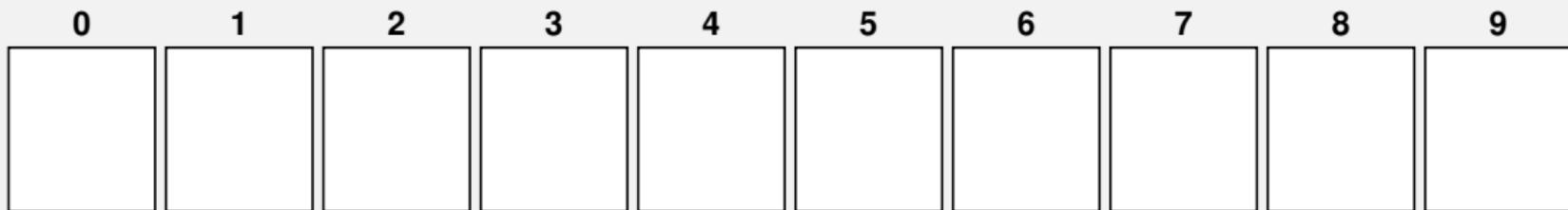
Speicher



Register

Euklid in der Box

Speicher



Links

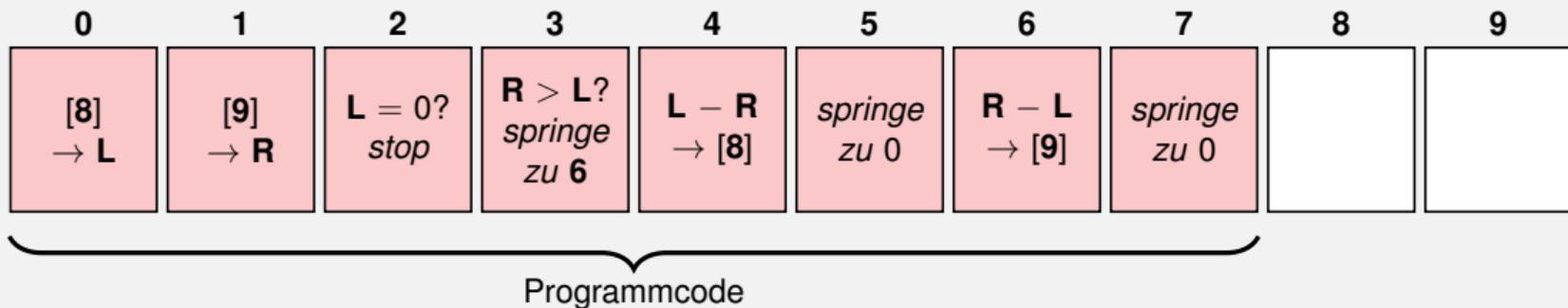
Rechts



Register

Euklid in der Box

Speicher



Links

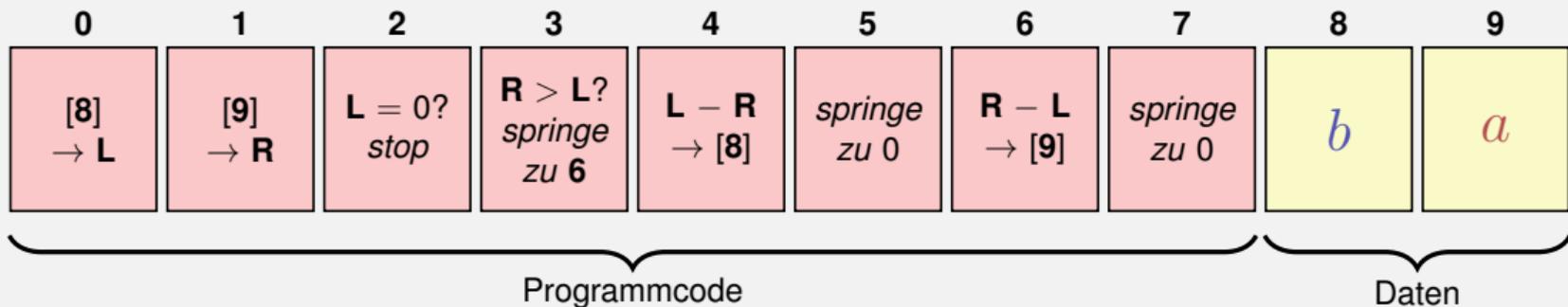
Rechts



Register

Euklid in der Box

Speicher



Links

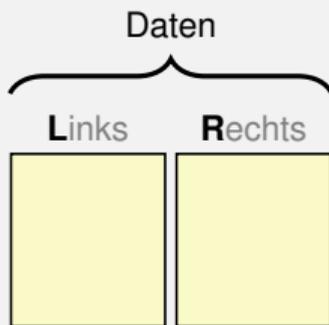
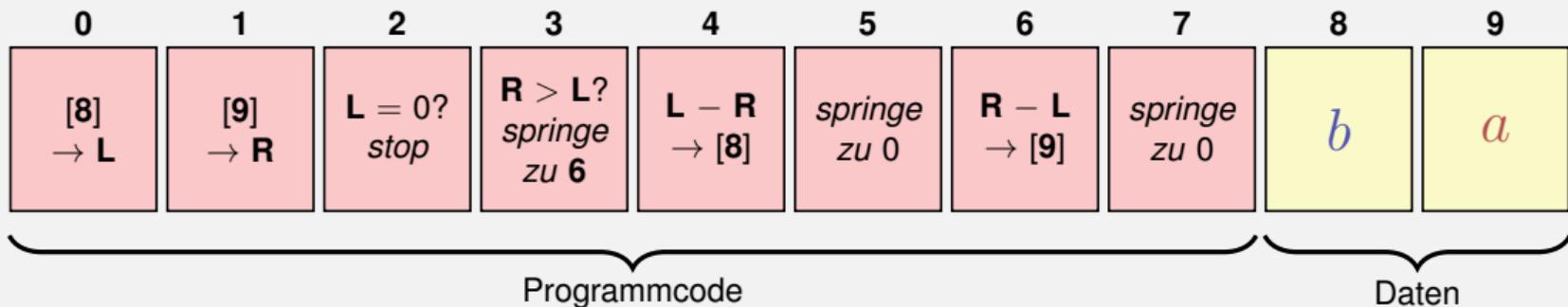
Rechts



Register

Euklid in der Box

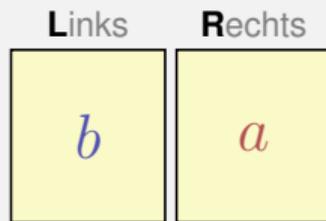
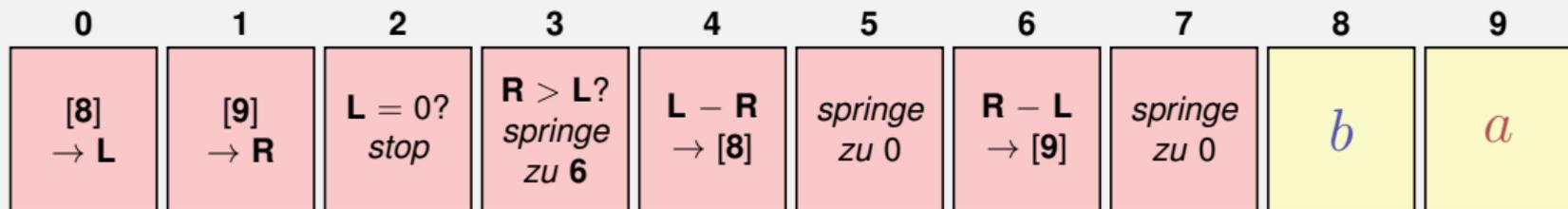
Speicher



Register

Euklid in der Box

Speicher



Register

Solange $b \neq 0$

Wenn $a > b$ dann

$$a \leftarrow a - b$$

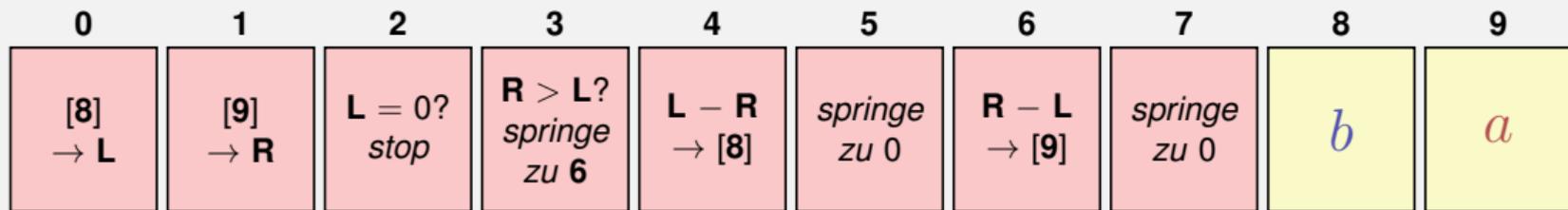
Sonst:

$$b \leftarrow b - a$$

Ergebnis: a .

Euklid in der Box

Speicher



Solange $b \neq 0$

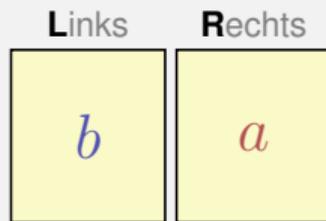
Wenn $a > b$ dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

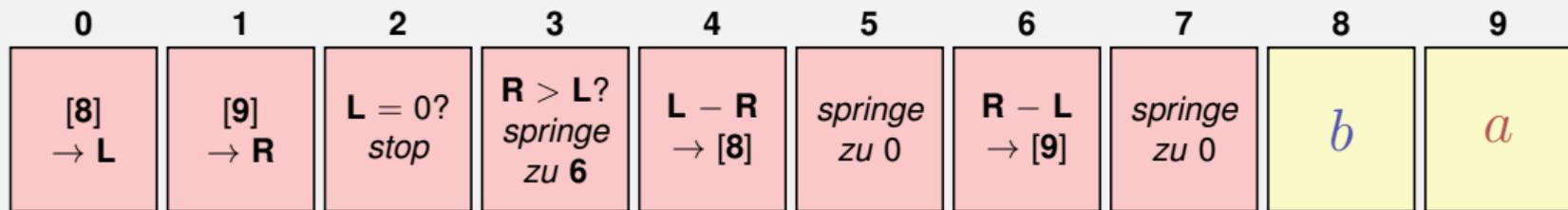
Ergebnis: a .



Register

Euklid in der Box

Speicher



Solange $b \neq 0$

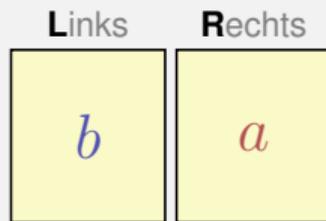
Wenn $a > b$ dann

$$a \leftarrow a - b$$

Sonst:

$$b \leftarrow b - a$$

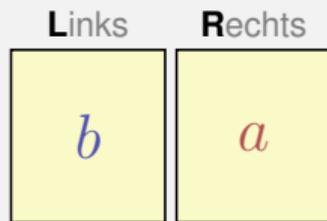
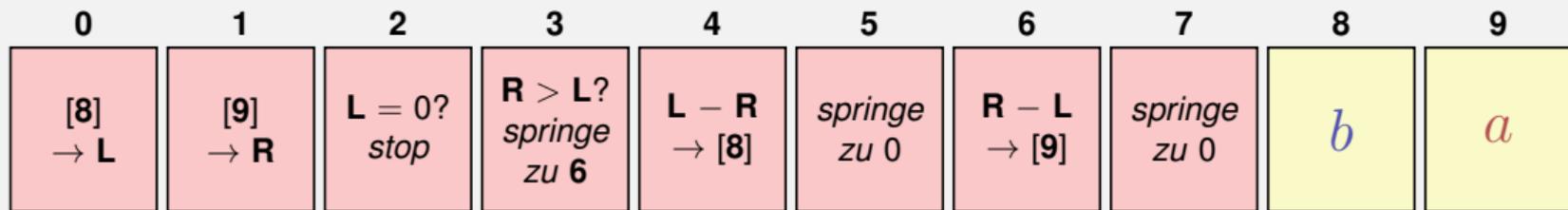
Ergebnis: a .



Register

Euklid in der Box

Speicher



Register

Solange $b \neq 0$

Wenn $a > b$ dann

$$a \leftarrow a - b$$

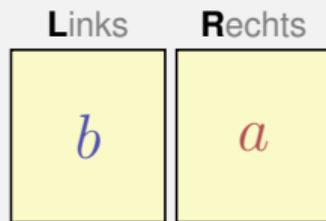
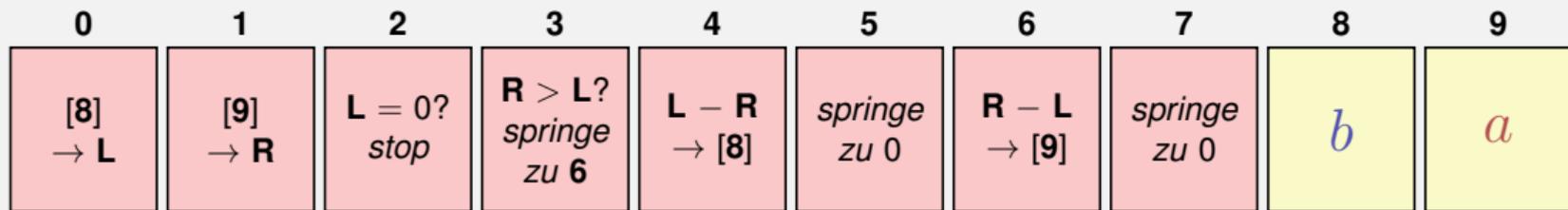
Sonst:

$$b \leftarrow b - a$$

Ergebnis: a .

Euklid in der Box

Speicher



Register

Solange $b \neq 0$

Wenn $a > b$ dann

$$a \leftarrow a - b$$

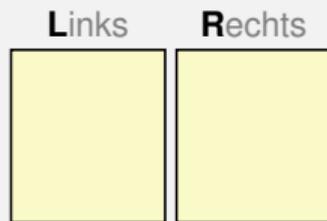
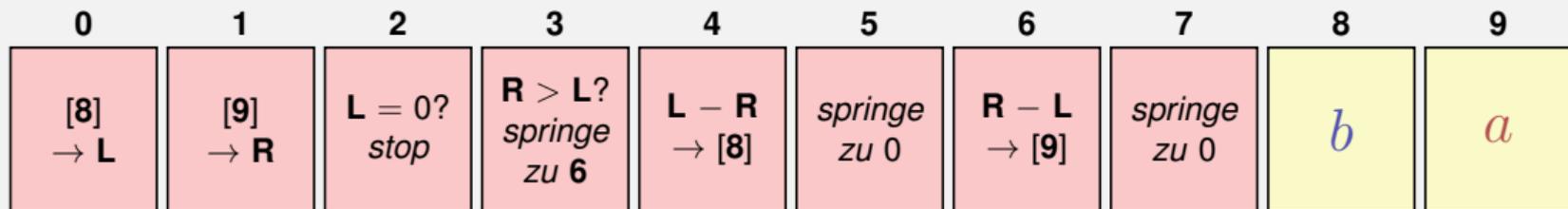
Sonst:

$$b \leftarrow b - a$$

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 .

ETH: Pionierin der modernen Informatik

1950: ETH mietet Konrad Zuses Z4, den damals einzigen funktionierenden Computer in Europa.

NEUE ZÜRCHER ZEITUNG

TECHNIK

Mittwoch, 20. August 1950 Blatt 25
Mittagsausgabe Nr. 276 (10)

Das programmgesteuerte Rechengesetz an der Eidgenössischen Technischen Hochschule in Zürich

Die Entwicklung programmgesteuerter Rechenmaschinen in den Vereinigten Staaten von Amerika wurde im letzten «Eidgenössischen Rechenmaschinen» (vgl. Nr. 218 der «N.Z.Z.» vom 23. Oktober 1948) und «Die neuen elektronischen Rechenmaschinen» (vgl. Nr. 272 der «N.Z.Z.» vom 26. April 1950) behandelt. Nächstebst soll von einem direkt deutschen Hersteller — Frau K. G. Zuse — die erste sein, welche im Juli dieses Jahres am Institut für angewandte Mathematik der Eidgenössischen Technischen Hochschule in Zürich, das unter der Leitung von Prof. Dr. E. Stiefel steht, in Betrieb genommen wurde. Damit ist diesem Institut in der Lage, dem in der Schweiz immer stärker werdenden Bedürfnis nach einer leistungsfähigen Rechenhilfe für numerische Rechnungen wenigstens teilweise gerecht zu werden. Besondere sind gewisse mathematische Probleme behandelbar worden, und die Erlösung vieler anderer Aufgaben ist vorhanden.

Merkmale des Gerätes

Das Gerät ist ein Glied in dem längeren Entwicklungsprogramm des letzten Konrad Zuse; es wurde im Auftrag des Institutes für angewandte Mathematik der E. T. H. unter Berücksichtigung von dessen Wünschen und Wünschen von Frau als «Modell Z 4» konstruiert. Die ursprüngliche Entwurfsarbeit an diesem Institut erfolgte in den Kriegsjahren und verlief völlig unabhängig von den Untersuchungen in den Vereinigten Staaten. Es ist diesem Institut festzustellen, was für die meisten weiteren mathematischen Probleme, bedingt durch gewisse Limitierungen, die sich aber während der ersten Phase manifestierten, die Lösung dieser Probleme im wesentlichen eine ganz selbstverständliche Fortsetzung beizubringen war.

Eine kurze technische Charakterisierung lautet wie folgt: Elektronenröhren arbeitendes Gerät mit 2200 Röhren 20 Schlüsselkontakte und einem Speicher für 64 Zyklen, welcher mit zweifachem, spezialisiertem Schlüsselwerk arbeitet; Verwendung des Dualsystems und der bildschirmartigen Darstellung; Multiplikation mit 23 Stellen; Programmsteuerung mit Hilfe zweier Lochkarten, auf die teilweise vorgedruckte werden kann; Einmalige von Zahlen durch eine Lochkarte oder durch einen Lochstreifen; Abgabe der Resultate durch Lochstreifen, Lochkarten oder Druckstreifen.

Das duale Zahlensystem

Allgemein sind programmgesteuerte Rechenanlagen häufig des dualen Zahlensystems angepasst, welches nur die zwei Zustände Null und 1 verwendet, während die letzte Dezimalstelle

Lesen wie eine Dezimalzahl von rechts nach links, so erhält sich das Gewicht von Stelle zu Stelle um den Faktor 10. Im Dualsystem ist ein ähnliches dieser Faktor 10 durch 2 zu ersetzen. Also bedeutet die (dezimale) Zahl 101,12 die Ausdrücke:

$$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 5$$

Die Zahl 1 wird in beiden Systemen gleich dargestellt. Im dualen System von dezimalen Zahlen bedeutet es immer, schreiben wir die Stelle 1 als 1. — Dagegen verhält sich die 2 ab, indem sie das 10 darstellt; dies das bedeutet, ist 2 = 10₂. Wenn eine Zahl (ohne Stellen nach dem Komma) mehr eine Null benötigt wird, so vergrößert sie sich um den Faktor 2 (und nicht, wie im Dezimalsystem, um den Faktor 10). Auf diese Weise kann ein 10 = 2 auf einfache Weise gebildet werden: 10 = 4, 100 = 8, 1000 = 16, usw.

Die Dualzahl 1101,12 bedeutet sich also:

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1 \cdot 8 + 0 + 2 + 0 = 10$$

Es ist analog sind etwaige Stellen nach dem Komma zu interpretieren; so wird 1,101,12 wie folgt interpretiert:

$$1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 1 + 0 + \frac{1}{2} + \frac{1}{4} = 1,375$$

Der große Vorteil, der das Dualsystem für Rechenoperationen so geeignet macht, nämlich die Reduktion der Anzahl der verwendeten Symbole auf nur zwei, wird allerdings durch einen Nachteil erkauft: Es braucht mehr Stellen, um eine bestimmte Zahl darzustellen. Die einfachste Zahl 5

Änderung des Stellenwert durchkomma ist weiter lösbar.

Die landläufige Darstellung bringt eine gewisse Komplexität der Rechenoperationen mit sich. So müssen bei einer Addition die beiden Summanden zunächst zu verschiedenen Stellen, die beiden Kommas verschieben, um sie in Hand einer einzigen Operation zu legen können, was am Hand eines Beispiels erläutert werden soll. Damit der Leser nicht durch die vielen Stellen verwirrt wird, ist das Beispiel im Dualsystem durchgeführt; doch wird dieses erörtern, daß das Gerät in Wirklichkeit mit dezimalen Zahlen rechnet.

Es soll also etwa addiert werden: 2,45078 × 10³ + 3,97012 × 10³ (Man beachte, daß die beiden Zahlen nicht getrennt 1 und 10 lauten, also das Komma nachheren Stellen rücken muß.) Von diesem die beiden Summanden «ausgerichtet» werden, d. h. die beiden Exponenten sind, einander gleich gemacht, und zwar erhält der kleinere Exponent den Wert des größeren. Die Zahlen lauten nun, häufig interessanter geschrieben und abgelesen, wie folgt:

$$\begin{aligned} 2,45078 \times 10^3 &= 2,45078 \times 10^3 \\ 3,97012 \times 10^3 &= 3,97012 \times 10^3 \end{aligned}$$

Es ist ersichtlich, daß bei der Ablesung der beiden Zahlen rechts über die Stellen abgemittelt werden müssen; denn wenn die Summanden abgelesen gegeben waren, so will sich das Resultat nicht mehr ablesen Stellen erhalten.



Abb. 2: Die Arbeit bei der Fertigung eines Rechenplans. Die Arbeiter in der Leuchte sind auch zu sehen.

Beifüge können «bedingt» gegeben werden, d. h. ihre Ausdringung wird von der Natur eines bestimmten Binäresystems abhängig gemacht. Erst dann werden die arithmetischen Operationen

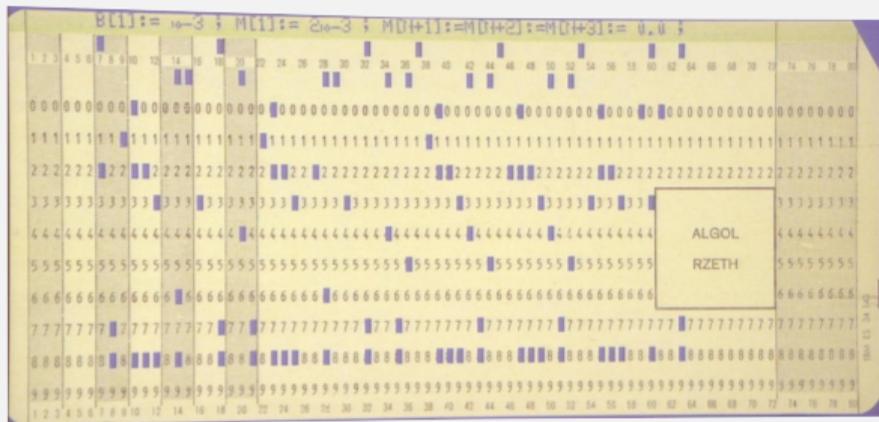
ETH: Pionierin der modernen Informatik

1956: Inbetriebnahme der ERMETH, entwickelt und gebaut an der ETH von Eduard Stiefel.



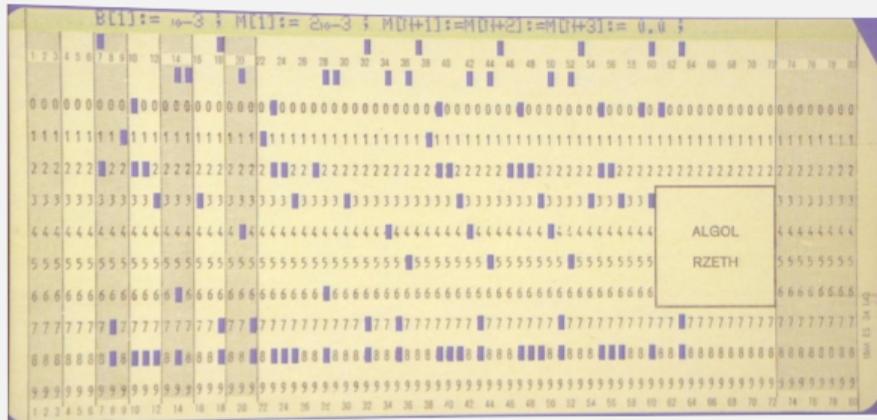
ETH: Pionierin der modernen Informatik

1958–1963: Entwicklung von ALGOL 60 (der ersten formal definierten Programmiersprache), unter anderem durch Heinz Rutishauer, ETH



ETH: Pionierin der modernen Informatik

1958–1963: Entwicklung von ALGOL 60 (der ersten formal definierten Programmiersprache), unter anderem durch Heinz Rutishauer, ETH



1964: Erstmals können ETH-Studierende selbst einen Computer programmieren (die CDC 1604, gebaut von Seymour Cray).

ETH: Pionierin der modernen Informatik



Die Klasse 1964 im Jahr 2015 (mit einigen Gästen)

ETH: Pionierin der modernen Informatik



Niklaus Wirth (* 1934)

ETH: Pionierin der modernen Informatik

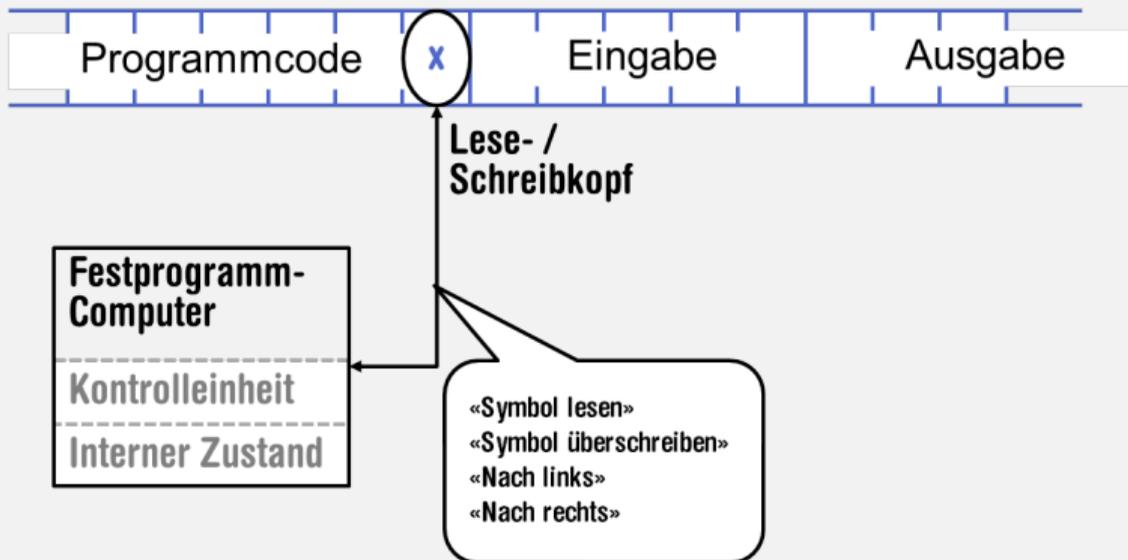
1968–1990: Niklaus Wirth entwickelt an der ETH die Programmiersprachen Pascal, Modula-2 und Oberon und 1980 die *Lilith*, einen der ersten Computer mit grafischer Benutzeroberfläche.



Computer – Konzept

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

Folge von Symbolen auf Ein- und Ausgabeband

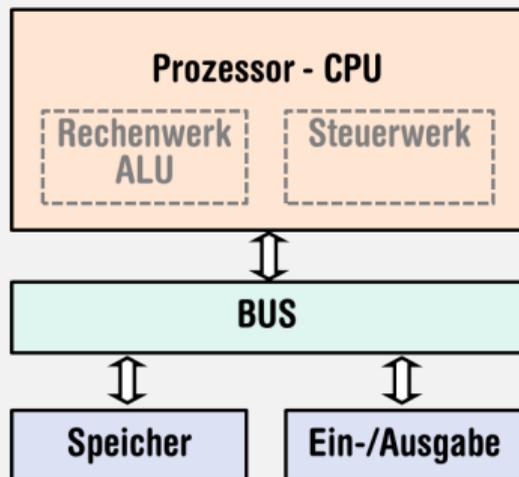


Alan Turing

Computer – Umsetzung

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

Von Neumann Architektur



Konrad Zuse



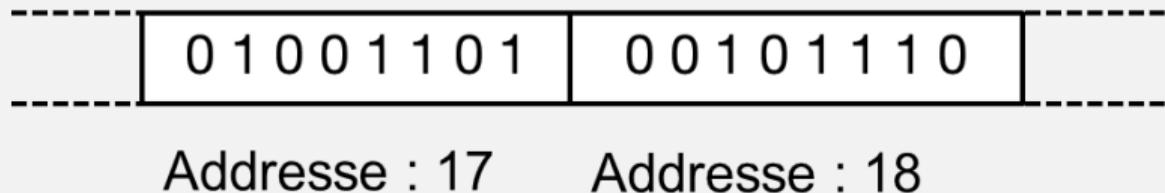
John von Neumann

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...

Rechengeschwindigkeit

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



30 m

arbeitet ein heutiger Desktop-PC mehr als 100

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.²

²Uniprozessor 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

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)
- Die meisten qualifizierten Jobs benötigen zumindest elementare Programmierkenntnisse.

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)
- Die meisten qualifizierten Jobs benötigen zumindest elementare Programmierkenntnisse.
- Programmieren macht Spass!

Programmiersprachen

- Sprache, die der Computer "verstehen", ist sehr primitiv (Maschinensprache).
- Einfache Operationen müssen in 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, Modula, Oberon, Python . . .

Warum C++?

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

- C++ ist relevant in der Praxis.
- Für das wissenschaftliche Rechnen (wie es in der Mathematik und Physik gebraucht wird), bietet C++ viele nützliche Konzepte.
- C++ ist weit verbreitet und “läuft überall”
- C++ ist standardisiert, d.h. es gibt ein “offizielles” C++.

Warum C++?

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

- C++ ist relevant in der Praxis.
- Für das wissenschaftliche Rechnen (wie es in der Mathematik und Physik gebraucht wird), bietet C++ viele nützliche Konzepte.
- C++ ist weit verbreitet und “läuft überall”
- C++ ist standardisiert, d.h. es gibt ein “offizielles” C++.
- Der Dozent mag C++.

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
```

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.

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?

Was braucht es zum Programmieren?

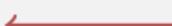
- **Editor:** Programm zum Ändern, Erfassen und Speichern von C++-Programmtext
- **Compiler:** Programm zum Übersetzen des Programmtexts in Maschinsprache

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;
}
```

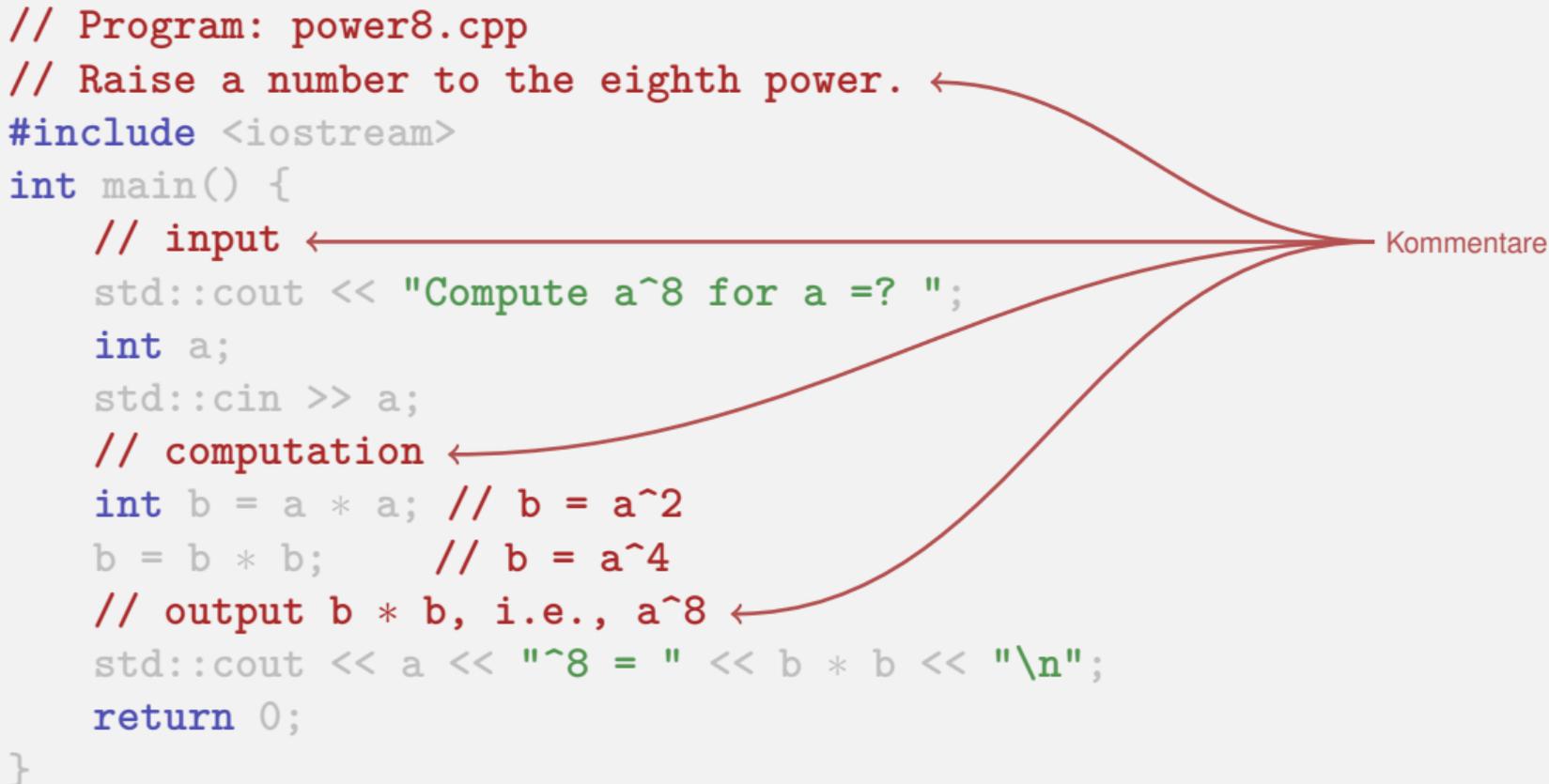
“Beiwerk”: Kommentare

```
// 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



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;}
```

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; ← Rückgabeanweisung  
}
```

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

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 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 concept of L-values and R-values in C++ with a code example. Red boxes highlight the expressions `"Compute a^8 for a =? "`, `b * b` (in the assignment), and `b * b` (in the output statement). Red arrows labeled "R-Wert" point from the text "R-Wert" to each of these three expressions, indicating that they are R-values (expressions that can be evaluated to a 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

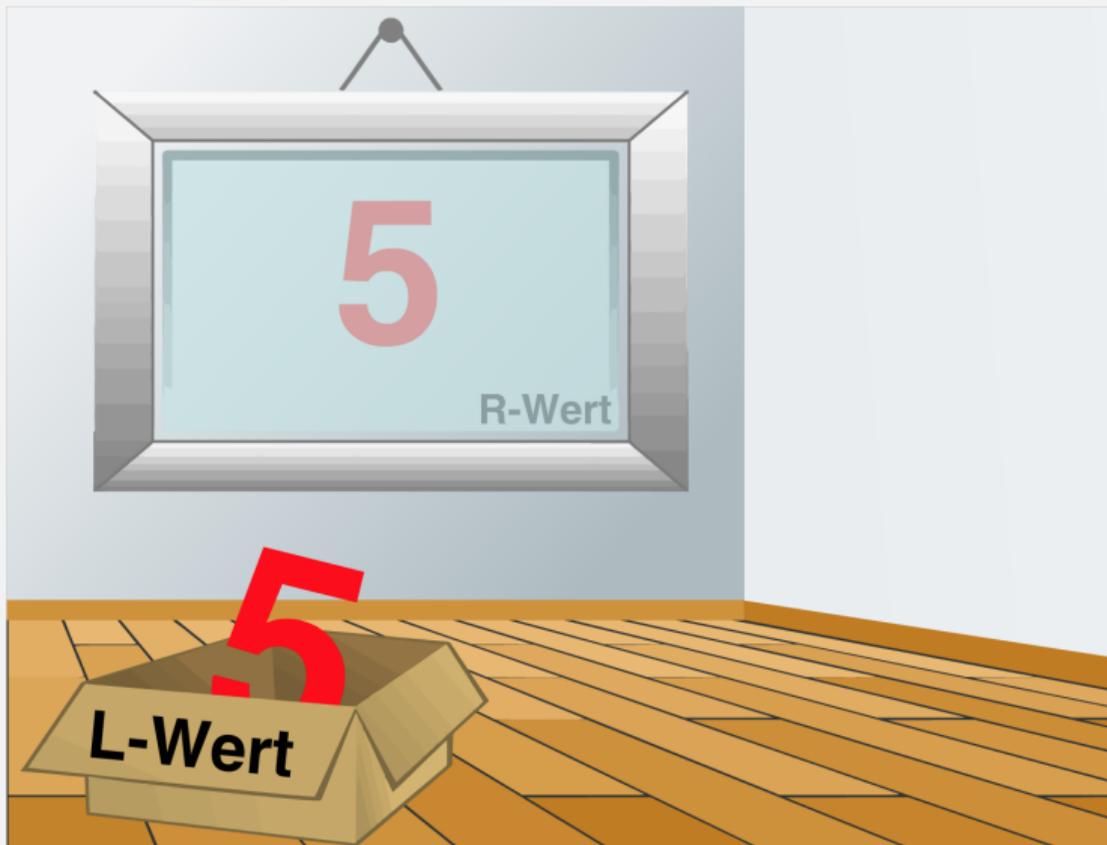
R-Wert (“**R**echts vom Zuweisungsoperator”)

- 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;
```

Diagram annotations:

- Linker Operand (Ausgabestrom) points to `std::cout`
- Ausgabe-Operator points to `<<`
- Rechter Operand (String) points to `"Compute a^8 for a =? "`

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

Zuweisungsoperator

Multiplikationsoperator