Informatik

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

Malte Schwerhoff, Felix Friedrich

HS 2018

Willkommen

zur Vorlesung Informatik

am MATH/PHYS Department der ETH Zürich.

Ort und Zeit:

Tuesday 13:15 - 15:00, ML D28, ML E12.
Pause 14:00 - 14:15, leichte Verschiebung möglich.

Vorlesungs-Webseite

`http://lec.inf.ethz.ch/ifmp`

Team

Chefassistent	Vytautas Astrauskas	
Back-Office	Inna Grijnevitch	
	Martin Clochard	
	Pavol Bielik	
Assistenten	Eliza Wszola	Moritz Schneider
	Alexander Hedges	Patrik Hadorn
	Viera Klasovita	Philippe Schlattner
	Max Egli	Yannik Ammann
	Christopher Lehner	Adrian Langenbach
	Orhan Saeedi	David Baur
	Maximillian Holst	Corminboeuf Etienne
	Benjamin Rothenberger	Tobias Klenze
	David Sommer	Sefidgar Seyed Reza
Dozenten	Dr. Malte Schwerhoff / Dr. Felix Friedrich	

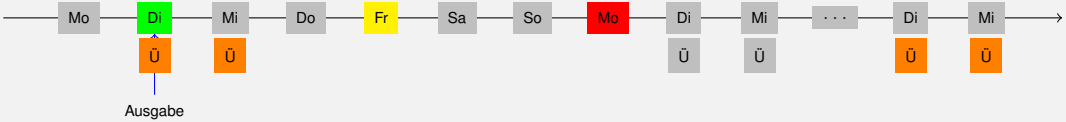
Einschreibung in Übungsgruppen

- Gruppeneinteilung selbstständig via Webseite
- Einschreibung breits offen

Einschreibung in Übungsgruppen

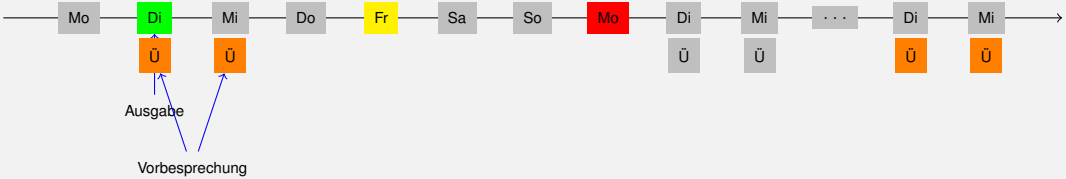
- Gruppeneinteilung selbstständig via Webseite
- Einschreibung breits offen
- 19 Gruppen insgesamt: 9 Dienstags 15-17 Uhr, 10 Mittwochs 10-12 Uhr
- 16 Gruppen auf Deutsch, 3 auf Englisch

Ablauf



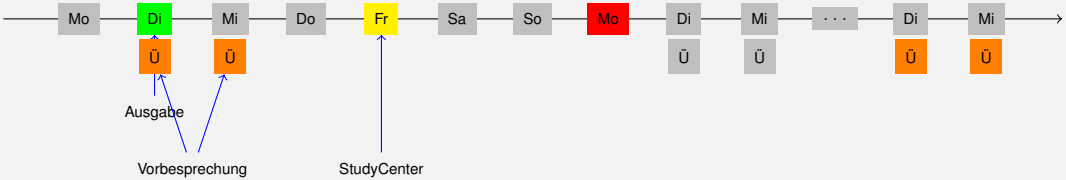
- Übungsblattausgabe zur Vorlesung (online)
- Vorbesprechung in der folgenden Übung (am selben/nächsten Tag)
- StudyCenter (`studycenter.ethz.ch`)
- Abgabe der Serie spätestens am Tag vor der nächsten Vorlesung (23:59h)
- Nachbesprechung der Serie in der übernächsten Übung. Feeback zu den Abgaben innerhalb einer Woche nach Abgabe.

Ablauf



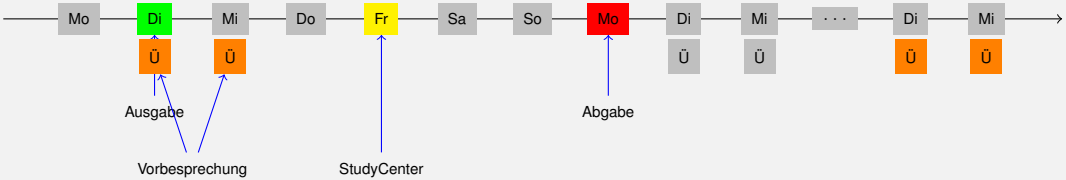
- Übungsblattausgabe zur Vorlesung (online)
- Vorbesprechung in der folgenden Übung (am selben/nächsten Tag)
- StudyCenter (`studycenter.ethz.ch`)
- Abgabe der Serie spätestens am Tag vor der nächsten Vorlesung (23:59h)
- Nachbesprechung der Serie in der übernächsten Übung. Feeback zu den Abgaben innerhalb einer Woche nach Abgabe.

Ablauf



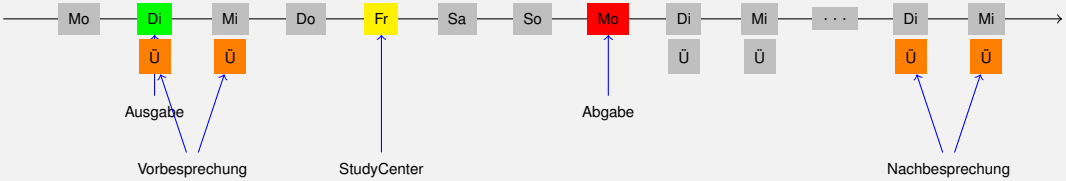
- Übungsblattausgabe zur Vorlesung (online)
- Vorbesprechung in der folgenden Übung (am selben/nächsten Tag)
- StudyCenter (`studycenter.ethz.ch`)
- Abgabe der Serie spätestens am Tag vor der nächsten Vorlesung (23:59h)
- Nachbesprechung der Serie in der übernächsten Übung. Feeback zu den Abgaben innerhalb einer Woche nach Abgabe.

Ablauf



- Übungsblattausgabe zur Vorlesung (online)
- Vorbesprechung in der folgenden Übung (am selben/nächsten Tag)
- StudyCenter (`studycenter.ethz.ch`)
- Abgabe der Serie spätestens am Tag vor der nächsten Vorlesung (23:59h)
- Nachbesprechung der Serie in der übernächsten Übung. Feeback zu den Abgaben innerhalb einer Woche nach Abgabe.

Ablauf



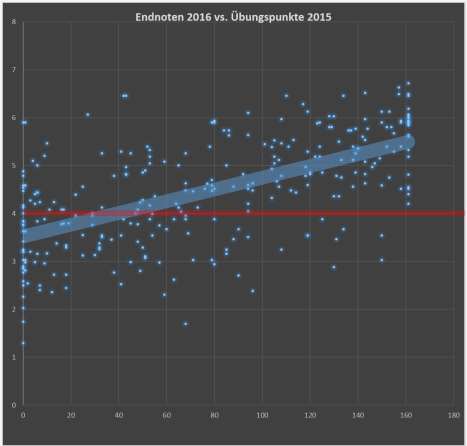
- Übungsblattausgabe zur Vorlesung (online)
- Vorbesprechung in der folgenden Übung (am selben/nächsten Tag)
- StudyCenter (`studycenter.ethz.ch`)
- Abgabe der Serie spätestens am Tag vor der nächsten Vorlesung (23:59h)
- Nachbesprechung der Serie in der übernächsten Übung. Feeback zu den Abgaben innerhalb einer Woche nach Abgabe.

Zu den Übungen

- Bearbeitung der wöchentlichen Uebungsserien ist also freiwillig, wird aber *dringend* empfohlen!

Zu den Übungen

- Bearbeitung der wöchentlichen Uebungsserien ist also freiwillig, wird aber *dringend* empfohlen!



Relevantes für die Prüfung

Prüfungsstoff für die Endprüfung (in der Prüfungssession 2018) schliesst ein

- Vorlesungsinhalt (Vorlesung, Handout) und
- Übungsinhalte (Übungsstunden, Übungsaufgaben).

Relevantes für die Prüfung

Prüfung ist schriftlich.

Es wird sowohl praktisches Wissen (Programmierfähigkeit) als auch theoretisches Wissen (Hintergründe, Systematik) geprüft.

Unser Angebot (VVZ)

- Ihre Programmierübungen werden (halb)automatisch bewertet. Durch Bearbeitung der wöchentlichen Übungsserien kann ein Bonus von maximal 0.25 Notenpunkten erarbeitet werden, der an die Prüfung mitgenommen wird.
- Der Bonus ist proportional zur erreichten Punktzahl von speziell markierten Bonusaufgaben, wobei volle Punktzahl einem Bonus von 0.25 entspricht. Die Zulassung zu speziell markierten Bonusaufgaben hängt von der erfolgreichen Absolvierung anderer Übungsaufgaben ab. Der erreichte Notenbonus verfällt, sobald die Vorlesung neu gelesen wird.

Unser Angebot (Konkret)

- Insgesamt 3 Bonusaufgaben; 2/3 der Punkte reichen für 0.25 Bonuspunkte für die Prüfung
- Sie können also z.B. 2 Bonusaufgaben zu 100% lösen, oder 3 Bonusaufgaben zu je 66%, oder ...
- Bonusaufgaben müssen durch erfolgreich gelöste Übungsserien freigeschaltet ($\rightarrow$ Experience Points) werden
- Es müssen wiederum nicht alle Übungsserien vollständig gelöst werden, um eine Bonusaufgabe freizuschalten
- Details: Kurswebseite, Übungsstunden, Online-Übungssystem (Code Expert)

Akademische Lauterkeit

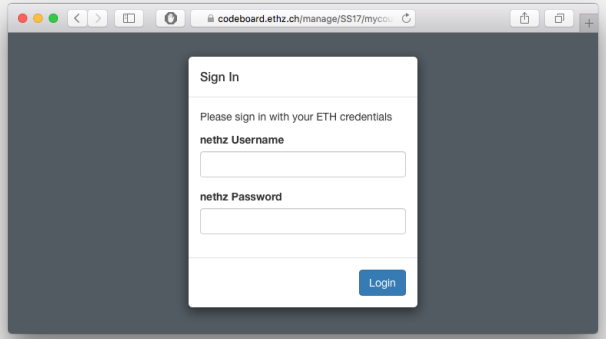
Regel: Sie geben nur eigene Lösungen ab, welche Sie selbst verfasst und verstanden haben.

Wir prüfen das (zum Teil automatisiert) nach und behalten uns insbesondere mündliche Prüfgespräche vor.

Sollten Sie zu einem Gespräche eingeladen werden: geraten Sie nicht in Panik. Es gilt primär die Unschuldsvermutung. Wir wollen wissen, ob Sie verstanden haben, was Sie abgegeben haben.

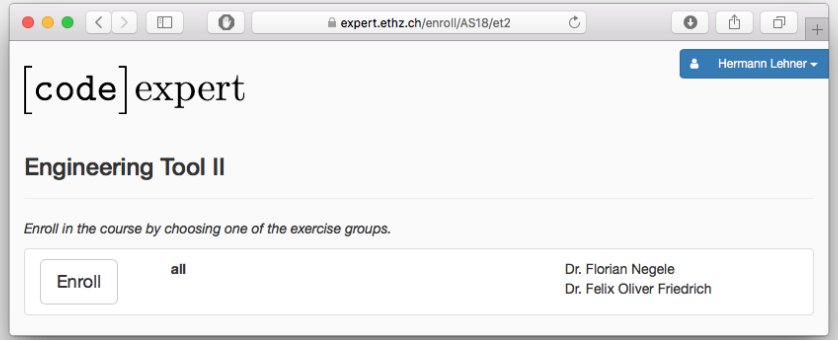
Einschreibung in Übungsgruppen - I

- Besuchen Sie `http://expert.ethz.ch/enroll/AS18/ifmp`
- Loggen Sie sich mit Ihrem nethz Account ein.

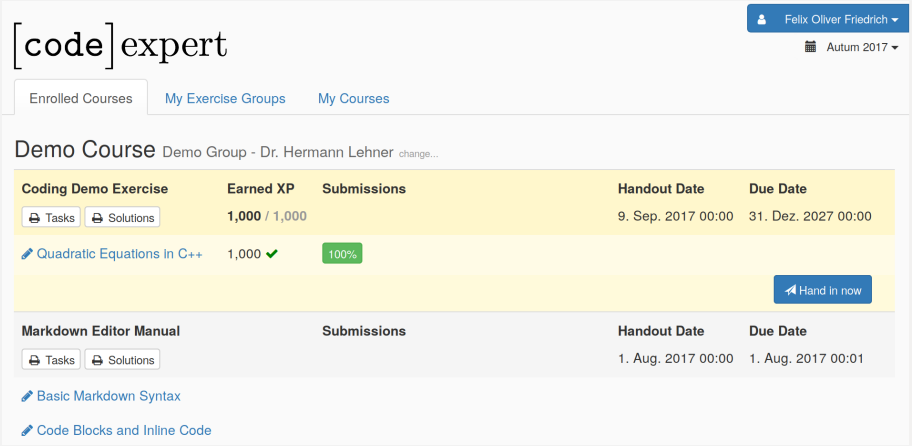


Einschreibung in Übungsgruppen - II

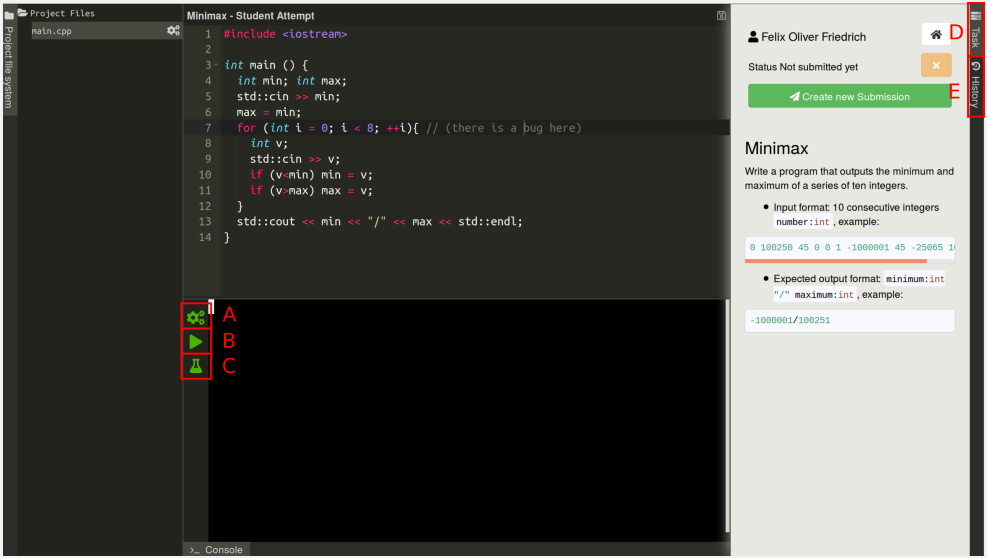
Schreiben Sie sich im folgenden Dialog in eine Übungsgruppe ein.



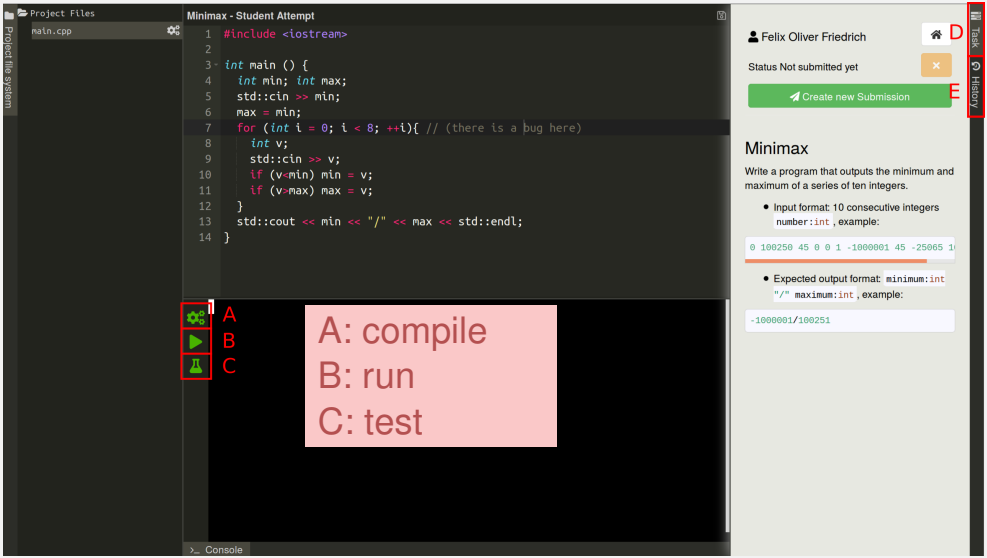
Übersicht


[code]expert
Felix Oliver Friedrich
Autum 2017
Enrolled Courses
My Exercise Groups
My Courses
Demo Course Demo Group - Dr. Hermann Lehner change...
Coding Demo Exercise
Earned XP
Submissions
Handout Date
Due Date
Tasks
Solutions
1,000 / 1,000
9. Sep. 2017 00:00
31. Dez. 2027 00:00
Quadratic Equations in C++
1,000
100%
Hand in now
Markdown Editor Manual
Submissions
Handout Date
Due Date
Tasks
Solutions
1. Aug. 2017 00:00
1. Aug. 2017 00:01
Basic Markdown Syntax
Code Blocks and Inline Code

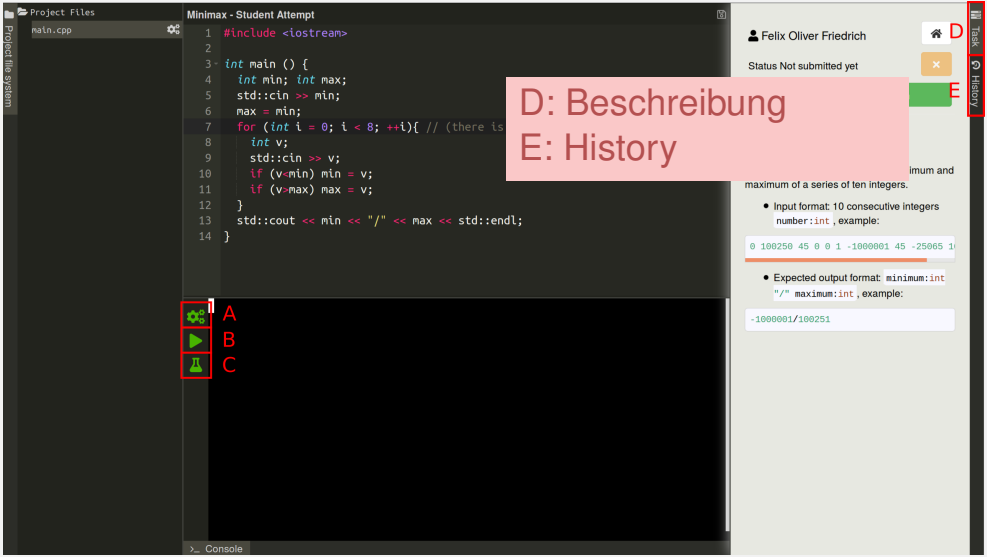
Programmierübung



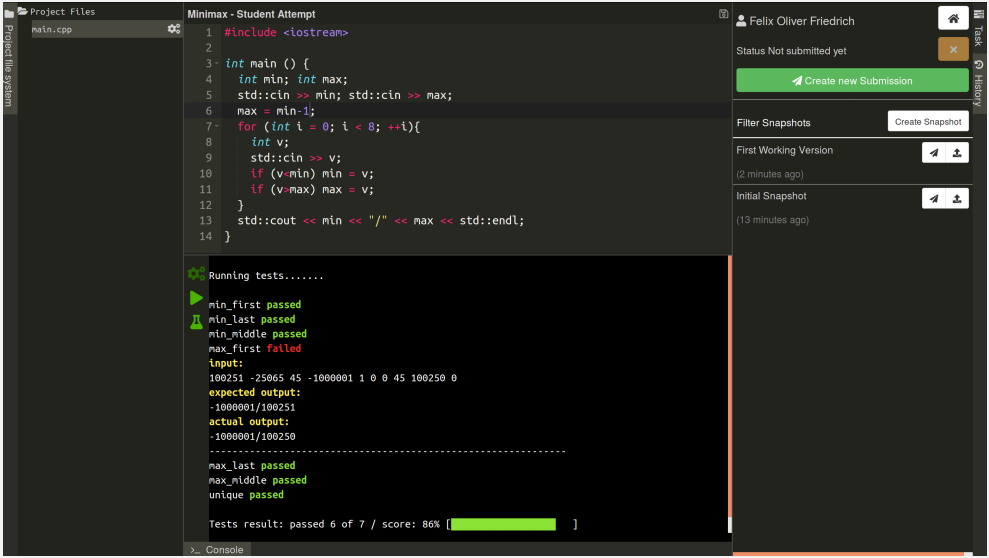
Programmierübung



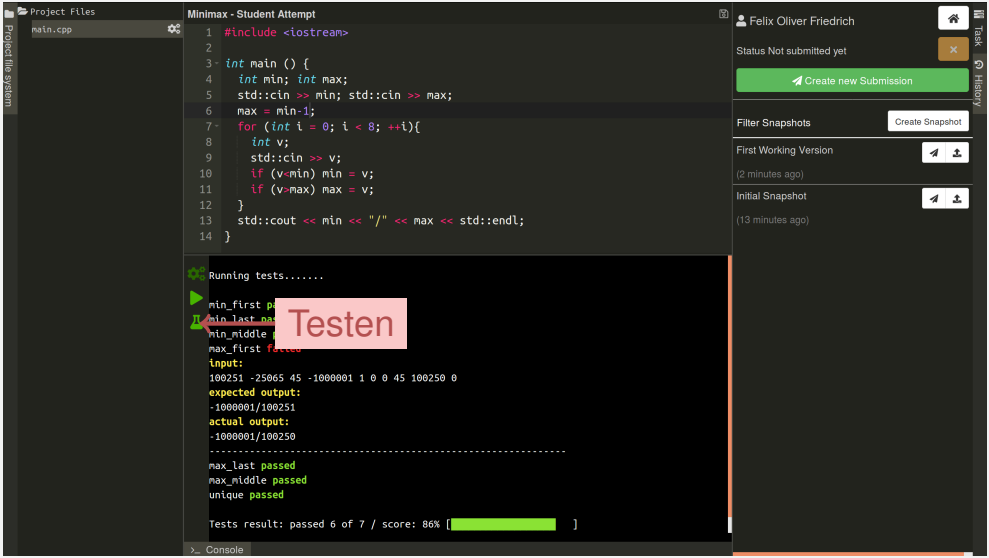
Programmierübung



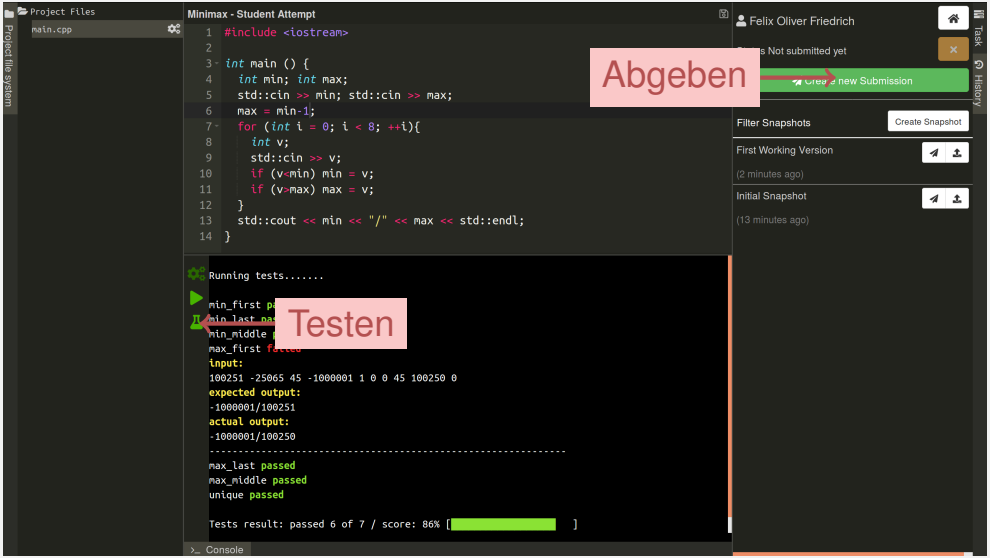
Testen und Abgeben



Testen und Abgeben



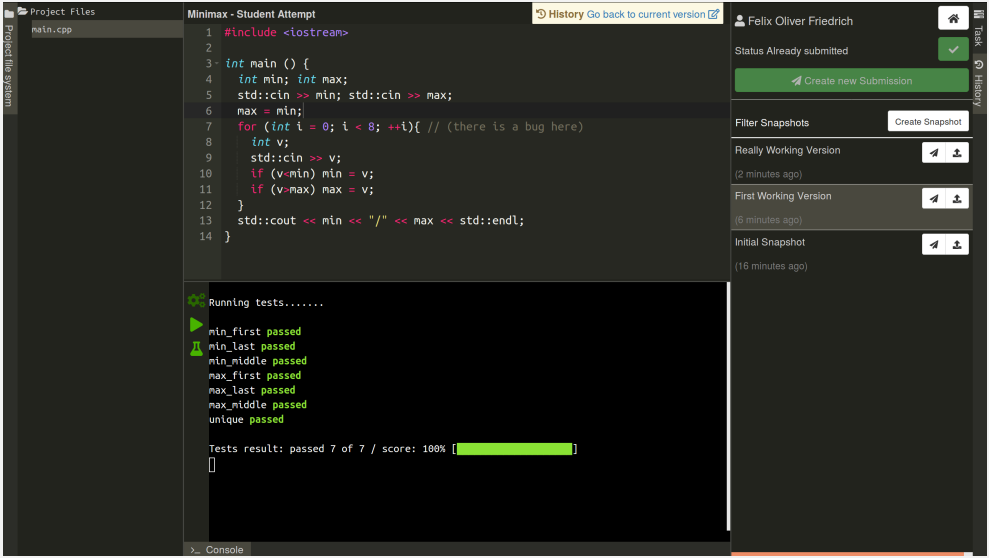
Testen und Abgeben



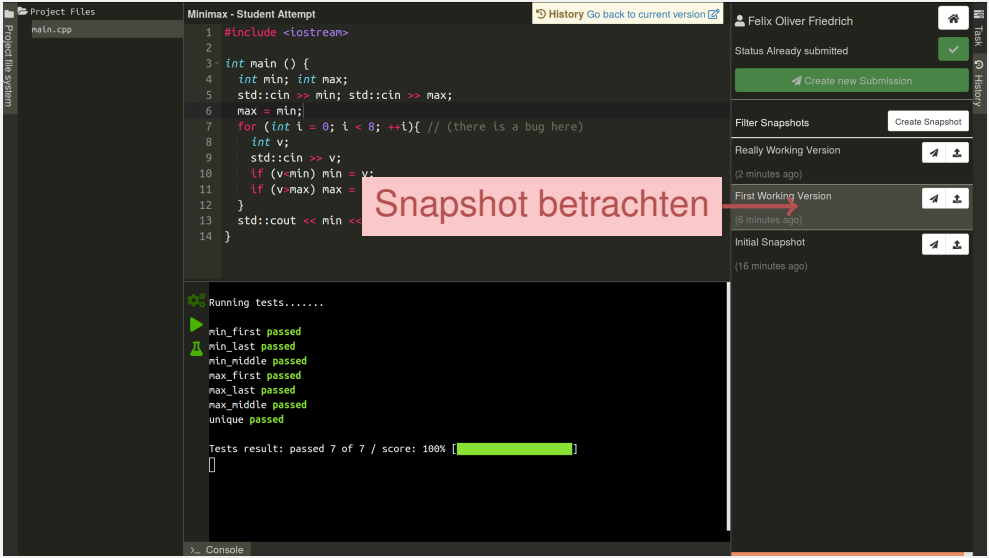
Wo ist der Save Knopf?

- Das Filesystem ist transaktionsbasiert und es wird laufend gespeichert („autosave“). Beim Öffnen eines Projektes findet man immer den zuletzt gesehenen Zustand wieder.
- Der derzeitige Stand kann als (benannter) *Snapshot* festgehalten werden. Zu gespeicherten Snapshots kann jederzeit zurückgekehrt werden.
- Der aktuelle Stand kann als Snapshot abgegeben (submitted) werden. Zudem kann jeder gespeicherts Snapshot abgegeben werden.

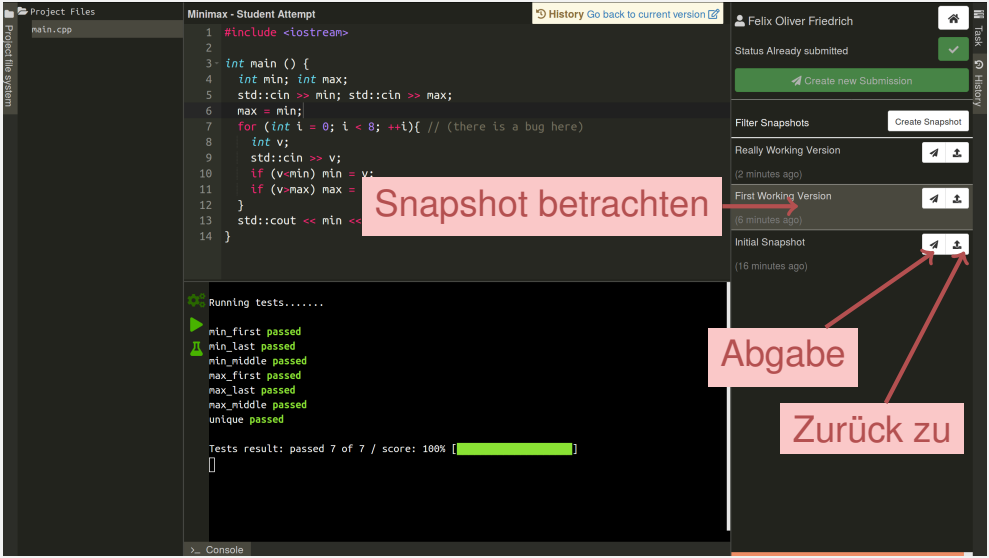
Snapshots


Project Files
main.cpp
Project file system
Minimax - Student Attempt
History Go back to current version
#include <iostream>
int main () {
int min; int max;
std::cin >> min; std::cin >> max;
max = min;
for (int i = 0; i < 8; ++i){ // (there is a bug here)
int v;
std::cin >> v;
if (v<min) min = v;
if (v>max) max = v;
}
std::cout << min << "/" << max << std::endl;
}
Running tests.......
min_first passed
min_last passed
min_middle passed
max_first passed
max_last passed
max_middle passed
unique passed
Tests result: passed 7 of 7 / score: 100% [
]
Console
Felix Oliver Friedrich
Status Already submitted
Create new Submission
Filter Snapshots
Create Snapshot
Really Working Version
(2 minutes ago)
First Working Version
(6 minutes ago)
Initial Snapshot
(16 minutes ago)
Task
History

Snapshots


Project Files
main.cpp
Project file system
Minimax - Student Attempt
History Go back to current version
#include <iostream>
int main () {
int min; int max;
std::cin >> min; std::cin >> max;
max = min;
for (int i = 0; i < 8; ++i){ // (there is a bug here)
int v;
std::cin >> v;
if (v<min) min = v;
if (v>max) max =
}
std::cout << min <<
}
Snapshot betrachten
Running tests.....
min_first passed
min_last passed
min_middle passed
max_first passed
max_last passed
max_middle passed
unique passed
Tests result: passed 7 of 7 / score: 100% [
Console
Felix Oliver Friedrich
Status Already submitted
Create new Submission
Filter Snapshots
Create Snapshot
Really Working Version
(2 minutes ago)
First Working Version
(6 minutes ago)
Initial Snapshot
(16 minutes ago)
Task
History

Snapshots


Project Files
main.cpp
Project file system
Minimax - Student Attempt
History Go back to current version
#include <iostream>
int main () {
int min; int max;
std::cin >> min; std::cin >> max;
max = min;
for (int i = 0; i < 8; ++i){ // (there is a bug here)
int v;
std::cin >> v;
if (v<min) min = v;
if (v>max) max =
}
std::cout << min <<
}
Running tests.....
min_first passed
min_last passed
min_middle passed
max_first passed
max_last passed
max_middle passed
unique passed
Tests result: passed 7 of 7 / score: 100% [
Console
Felix Oliver Friedrich
Status Already submitted
Create new Submission
Filter Snapshots
Create Snapshot
Really Working Version
(2 minutes ago)
First Working Version
(6 minutes ago)
Initial Snapshot
(16 minutes ago)
Task
History
Snapshot betrachten
Abgabe
Zurück zu

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?

ETH: Pionierin der modernen Informatik

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

NEUE ZÜRCHER ZEITUNG

TECHNIK

Mittwoch, 30. August 1950 Blatt 15
Mittagausgabe Nr. 1796 (50)

Das programmgesteuerte Rechengerät an der Eidgenössischen Technischen Hochschule in Zürich

Die Entwicklung programmgesteuerter Rechenmaschinen in den Vereinigten Staaten von Amerika wurde in den Artikeln „Elektronische Rechenmaschinen“ (vgl. Nr. 2140 der „N. Z. Z.“ vom 13. Oktober 1948) und „Die neueste elektronische Rechenmaschine“ (vgl. Nr. 871 der „N. Z. Z.“ vom 26. April 1950) behandelt. Nachstehend soll von einem Gerät deutscher Herkunft — Zuse K.-G., Neukirchen — die Rede sein, welches 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 dieses Institut in der Lage, dem in der Schweiz immer stärker werdenden Bedürfnis nach einer leistungsfähigen Zentralstelle für numerische Rechnungen wenigstens teilweise gerecht zu werden. Bereits sind einige mathematische Probleme behandelt worden, und die Erledigung vieler anderer Aufgaben ist vorbereitet.

Merkmale des Gerätes

Das Gerät ist ein Glied in dem längeren Entwicklungsprogramm des Ingenieurs *Konrad Zuse;* es wurde im Auftrag des *Institutes für angewandte Mathematik* der E. T. H. unter Berücksichtigung von dessen Wünschen und Ideen von Zuse als *„Modell Z 4“* konstruiert. Die ursprüngliche Entwicklung in Deutschland erfolgte in den Kriegsjahren und verlief völlig unabhängig von den Untersuchungen in den Vereinigten Staaten. Es ist überaus interessant festzustellen, wie für die meisten wichtigen funktionellen Probleme beiderorts genau dieselbe Lösung gefunden wurde, wie aber anderseits gewissen Fragen sekundärer Wichtigkeit eine ganz unterschiedliche Bedeutung beigemessen wurde.

Eine kurze technische Charakterisierung lautet wie folgt: Elektromechanisch arbeitendes Gerät mit 2200 Relais, 21 Schrittschaltern und einem Speicher für 64 Zahlen, welcher mit neuartigen, mechanischen Schaltgliedern arbeitet; Verwendung des Dualsystems und der halblogarithmischen Darstellung; Multiplikationszeit 2,5 Sekunden; Programmsteuerung mit Hilfe zweier Lochstreifen, auf die wahlweise umgeschaltet werden kann; Eingabe von Zahlen durch eine Tastatur oder durch einen Lochstreifen; Abgabe der Resultate durch Lampenfeld, Lochstreifen oder Druckwerk.

Das duale Zahlsystem

Allgemein wird programmgesteuerten Rechengeräten häufig das duale Zahlsystem zugrunde gelegt, welches nur die zwei Zahlsymbole 0 und 1 verwendet, während das bekannte Dezimalsystem

Lesen wir eine Dezimalzahl von rechts nach links, so erhöht sich das Gewicht von Stelle zu Stelle um den Faktor 10. Im *Dualsystem* ist nun einfach dieser Faktor 10 durch 2 zu ersetzen. Also bedeutet die (nunmehr duale) Zahl abcd,efg den Ausdruck:

$$a \cdot 2^3 + b \cdot 2^2 + c \cdot 2^1 + d \cdot 2^0 + e \cdot 2^{-1} + f \cdot 2^{-2} + g \cdot 2^{-3}$$

Die Zahl 1 wird in beiden Systemen gleich dargestellt. Um jedoch duale von dezimalen Zahlen deutlich zu trennen, schreiben wir die duale 1 als L. — Dagegen weicht schon die 2 ab, indem sie dual L0 lautet; denn dies bedeutet ja $1 \cdot 2^1 + 0 \cdot 2^0 = 2$. Wenn einer Zahl (ohne Stellen nach dem Komma) rechts eine Null zugefügt wird, so vergrößert sie sich um den Faktor 2 (und nicht, wie im Dezimalsystem, um den Faktor 10). Auf diese Weise kann aus L0 = 2 auf einfachste Weise gebildet werden: L00 = 4, L000 = 8, L0000 = 16, usw.

Die Dualzahl L0L0L bedeutet nun also:

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 21$$

Ganz analog sind etwaige Stellen nach dem Komma zu interpretieren; so wird L, 0LL wie folgt übersetzt:

$$1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 1 + 1/4 + 1/8 = 1,375$$

Der große Vorteil, der das Dualsystem für Rechenautomaten 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 einstellige Zahl 8

Aenderung des Maßstabes durchgerechnet werden können.

Die beschriebene Darstellung bringt eine gewisse Komplikation der Rechenoperationen mit sich. So müssen vor einer Addition die beiden Summanden zunächst so verschoben werden, daß ihre Kommata untereinander zu liegen kommen, was an Hand eines Beispiels erläutert werden soll. Damit der Leser nicht durch das ungewohnte duale Zahlsystem verwirrt wird, ist das Beispiel im Dezimalsystem durchgeführt; doch wird daran erinnert, daß das Gerät in Wirklichkeit mit dualen Zahlen rechnet.

Es soll also etwa addiert werden: $2,345678 \times 10^2 + 9,876543 \times 10^{-1}$ (Man beachte, daß die eigentliche Zahl stets zwischen 1 und 10 liegt, also das Komma nach der ersten Stelle hat.) Nun müssen die beiden Summanden „ausgerichtet“ werden, d. h. die beiden Exponenten sind einander gleich zu machen, und zwar erhält der kleinere Exponent den Wert des größeren, also 2. Die Zahlen lauten nun, richtig untereinander geschrieben und addiert, wie folgt:

$$\begin{array}{r} 2,345678 \times 10^2 \\ 0,009876 \times 10^2 \\ \hline 2,355554 \times 10^2 \end{array}$$

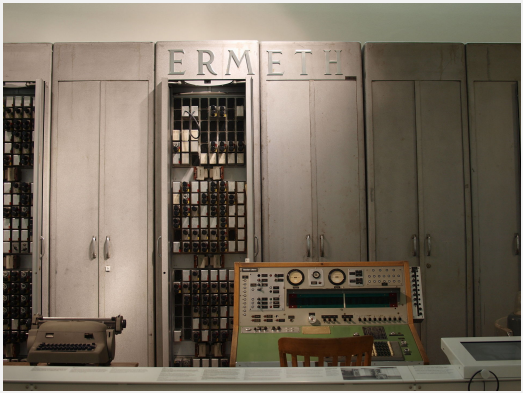
Es ist ersichtlich, daß bei der kleineren der beiden Zahlen rechts einige Stellen abgeschnitten werden mußten; denn wenn die Summanden siebenstellig gegeben waren, so soll auch das Resultat nicht mehr als sieben Stellen enthalten.

Abb. 2. Der Schaltpult bei der Fertigung eines Rechenplanes. Die Abtaster für den Lochstreifen sind deutlich sichtbar.

Befehle können „bedingt“ gegeben werden, d. h. ihre Ausführung wird von der Natur eines errechneten Resultates abhängig gemacht. Erst dadurch werden die außerordentlich weiten Perspek-

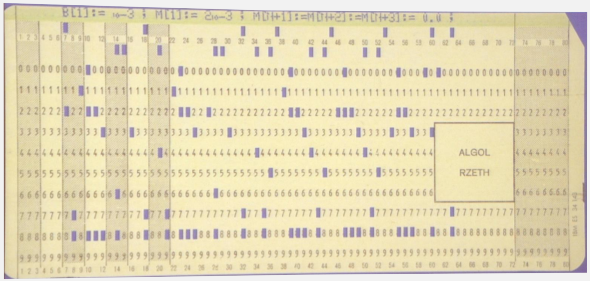
ETH: Pionierin der modernen Informatik

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



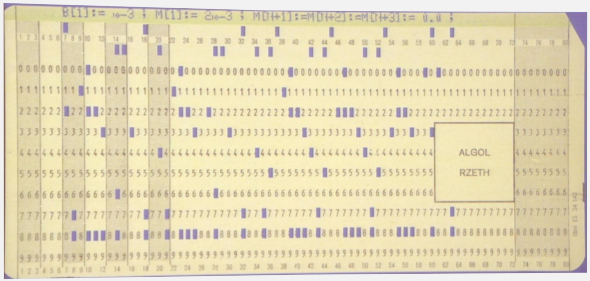
ETH: Pionierin der modernen Informatik

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.

Zurück in die Gegenwart: Inhalt dieser Vorlesung

- Systematisches Problemlösen mit Algorithmen und der Programmiersprache $\mathrm{C}{+}{+}$.
- 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)

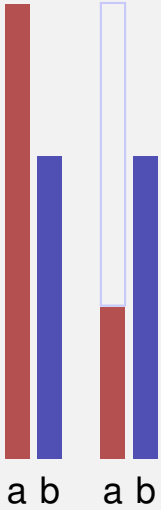
Der älteste nichttriviale Algorithmus

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

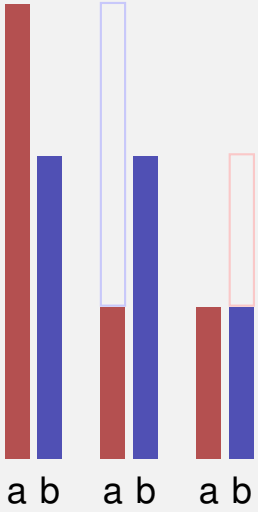
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



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

Der älteste nichttriviale Algorithmus

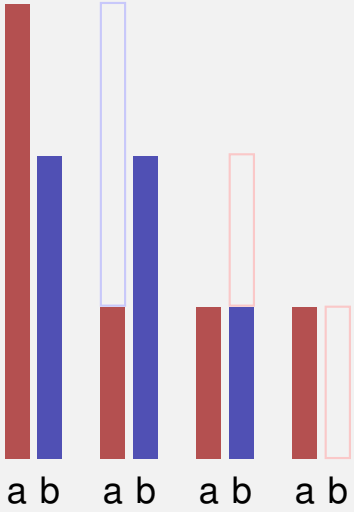
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



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

Der älteste nichttriviale Algorithmus

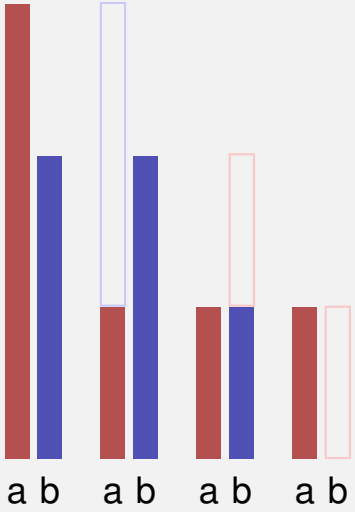
Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)



- 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)

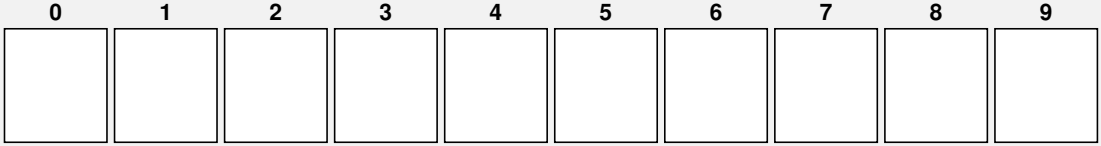
Algorithmen: 3 Abstraktionsstufen

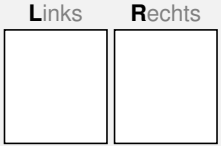
1. **Kernidee** (abstrakt)**:**
   Die Essenz eines Algorithmus' („Heureka-Moment“)
2. **Pseudocode** (semi-detailliert)**:**
   Für Menschen gemacht (Bildung, Korrektheit- und Effizienzdiskussionen, Beweise)
3. **Implementierung** (sehr detailliert)**:**
   Für Mensch & Computer gemacht (les- & ausführbar,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

Euklid: Kernidee und Pseudocode gesehen, Implementierung noch nicht

Euklid in der Box

Speicher

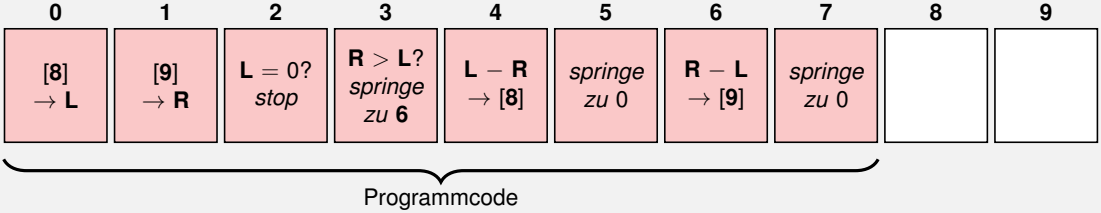


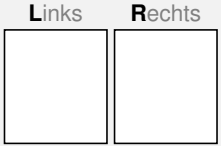


Register

Euklid in der Box

Speicher

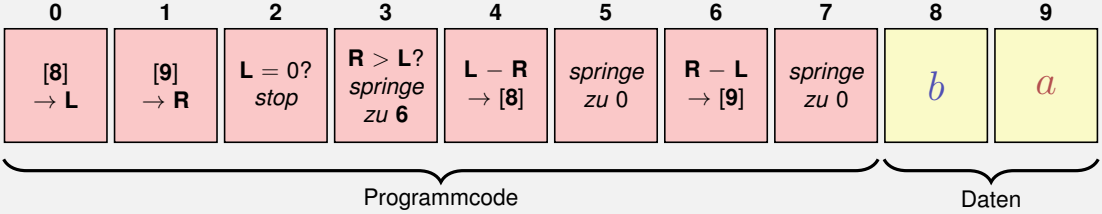


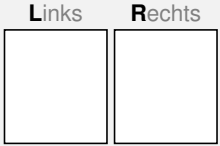


Register

Euklid in der Box

Speicher

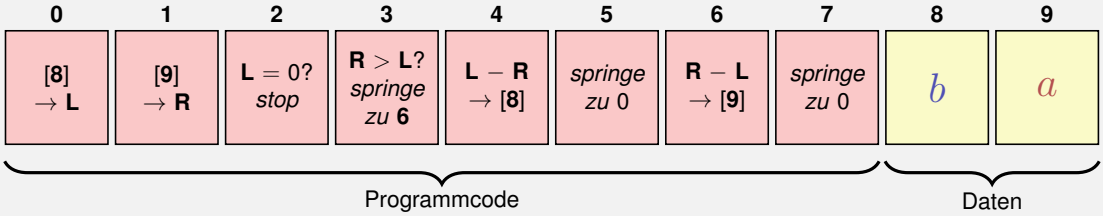


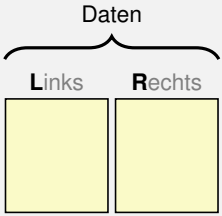


Register

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

$a \leftarrow a - b$

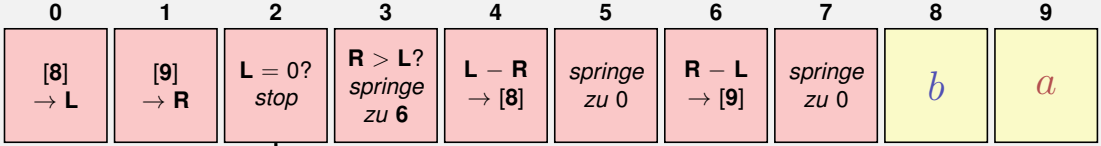
Sonst:

$b \leftarrow b - a$

Ergebnis: a.

Euklid in der Box

Speicher



Solange $b \neq 0$

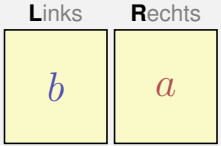
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

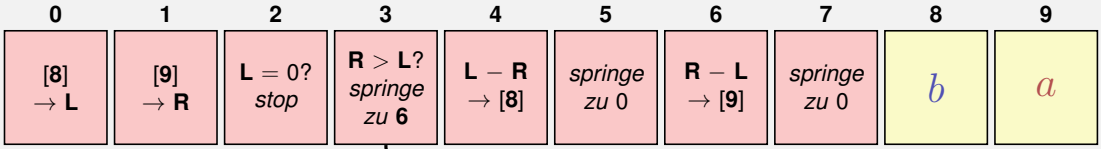
Ergebnis: a.



Register

Euklid in der Box

Speicher



Solange $b \neq 0$

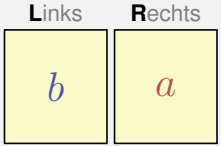
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

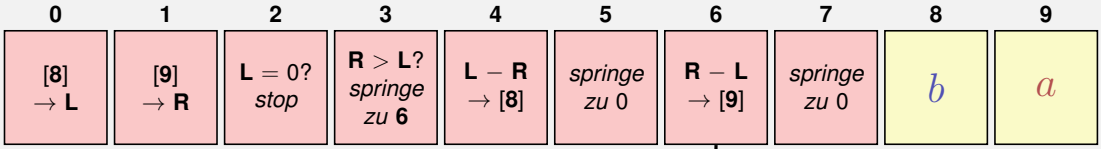
Ergebnis: a.



Register

Euklid in der Box

Speicher



Solange $b \neq 0$

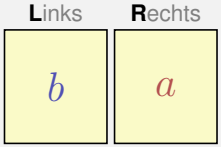
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

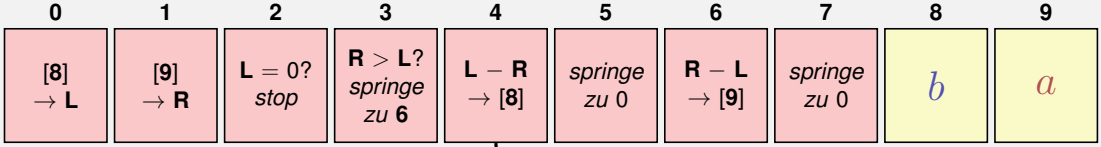
Ergebnis: a.



Register

Euklid in der Box

Speicher



Solange $b \neq 0$

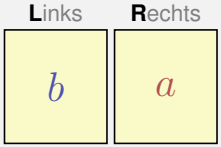
Wenn $a > b$ dann

$a \leftarrow a - b$

Sonst:

$b \leftarrow b - a$

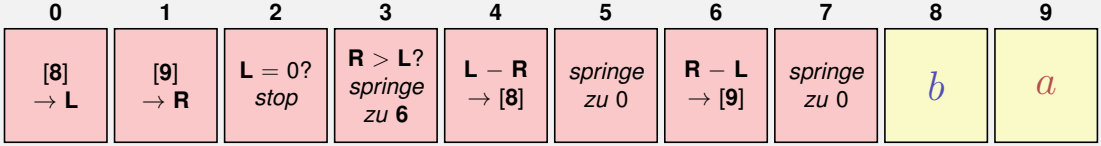
Ergebnis: a.

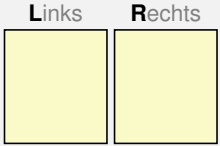


Register

Euklid in der Box

Speicher





Register

Solange $b \neq 0$

Wenn $a > b$ dann

$a \leftarrow a - b$

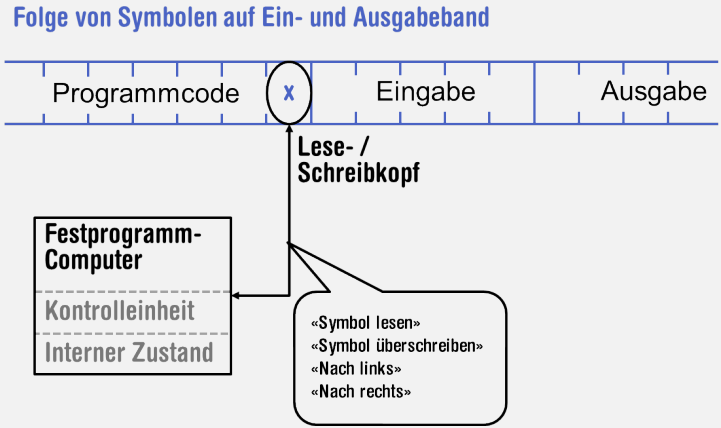
Sonst:

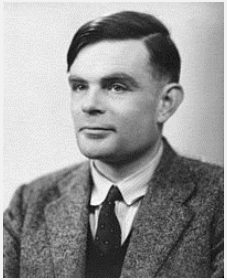
$b \leftarrow b - a$

Ergebnis: a.

Computer – Konzept

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





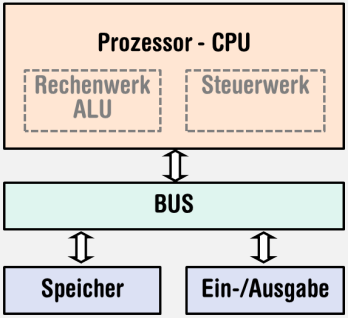
Alan Turing

Computer – Umsetzung

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

Konrad Zuse

Von Neumann Architektur



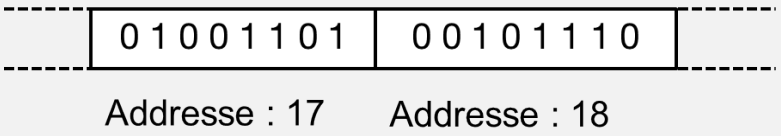
John von Neumann

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

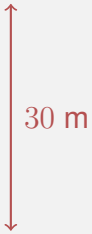
Rechengeschwindigkeit

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, 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)
- 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 (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, ...

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

Warum C++?

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

- C++ ist relevant in der Praxis (weit verbreitet) und „läuft überall“
- Für das wissenschaftliche Rechnen (wie es in der Mathematik und Physik gebraucht wird), bietet C++ viele nützliche Konzepte.
- C++ ist standardisiert, d.h. es gibt ein „offizielles“ C++.
- C++ ist eine der „schnellsten“ Programmiersprachen
- C++ eignet sich gut für Systemprogrammierung da es einen sorgfältigen Umgang mit Ressourcen (Speicher, ...) erlaubt/verlangt

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;
```

```
// 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;
```

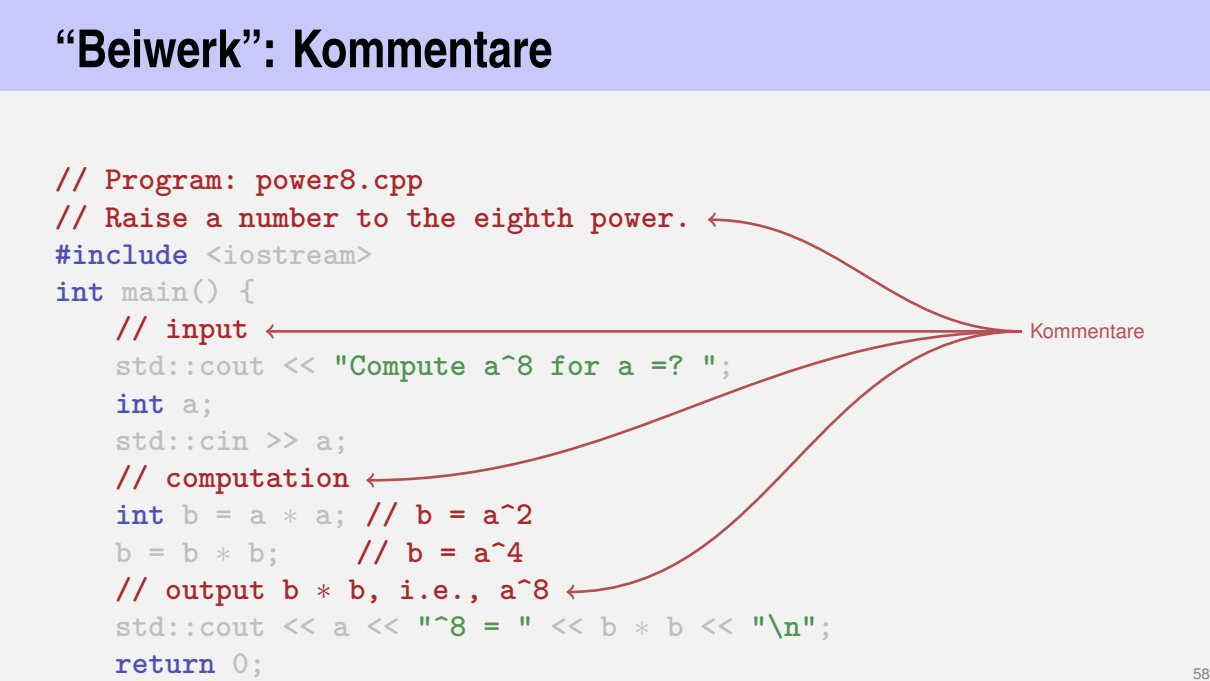
```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
int main() {
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation
    int b = a * a; // b = a^2 ←——— Berechne einen Wert ($a^2$)!
    b = b * b;     // b = a^4
    // 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 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;
```

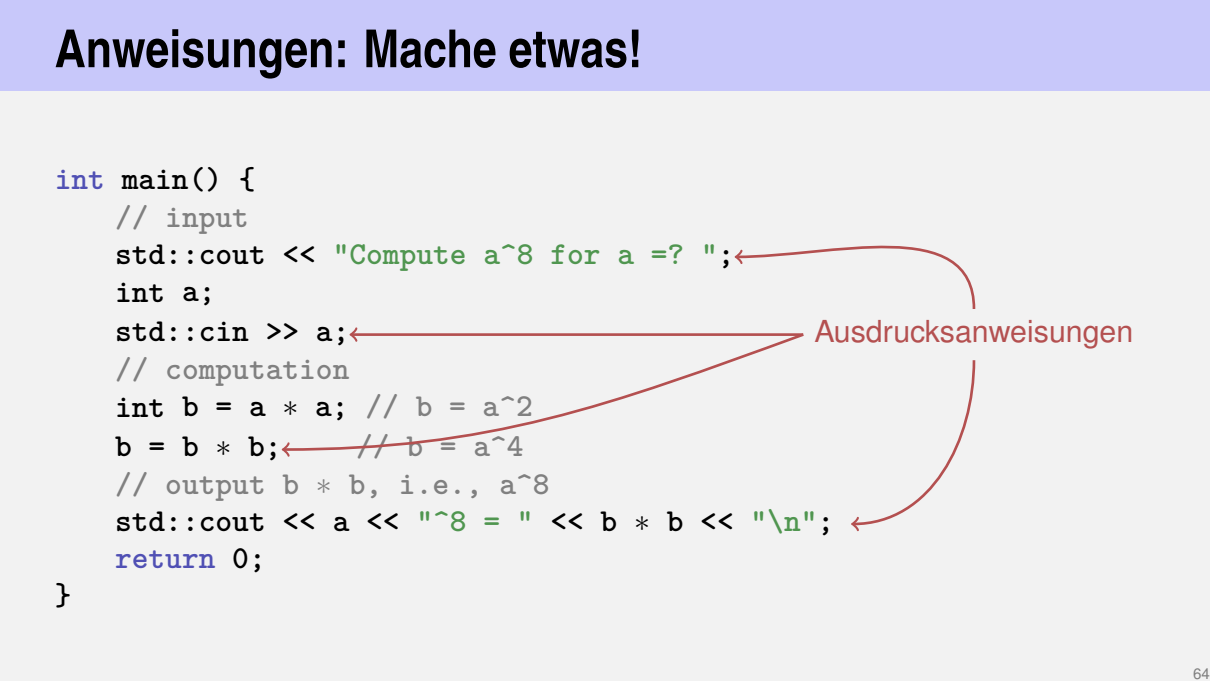
Funktionsdeklaration der main-Funktion

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*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;
    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

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
- Addresse: 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

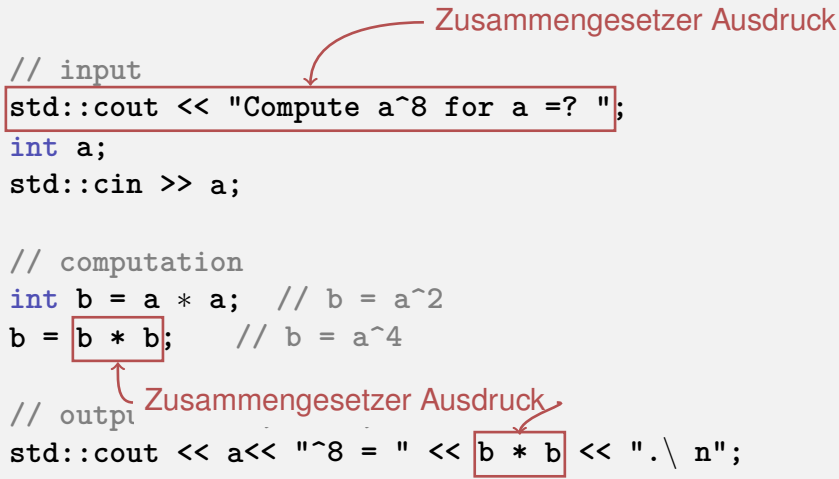
```
// input
std::cout << "Compute a^8 for a =? ";
int a;
std::cin >> a;
```

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;
```


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

← Zweifach zusammengesetzter Ausdruck

↑ 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

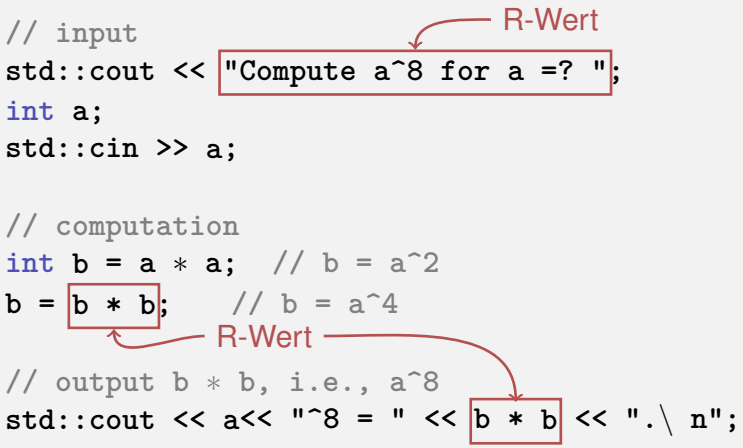
```
// input
std::cout << "Compute a^8 for a =? ";
int a;
std::cin >> a;

// 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



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

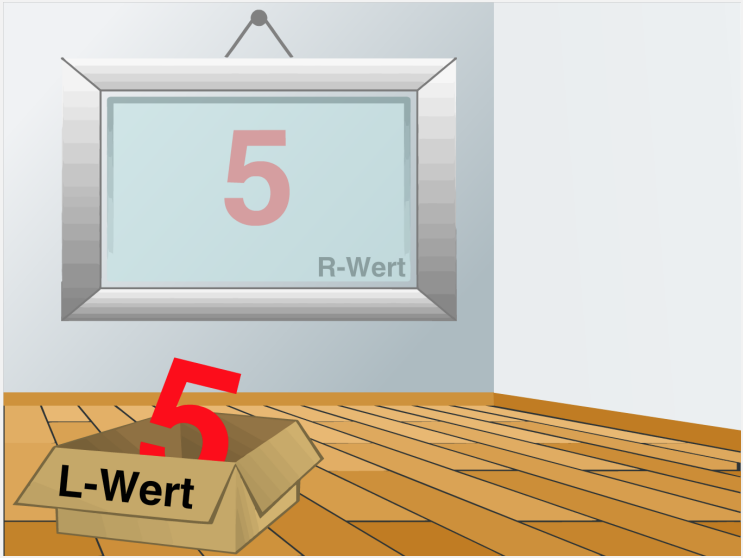
R-Wert (“**R**echts vom Zuweisungsoperator”)

- 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

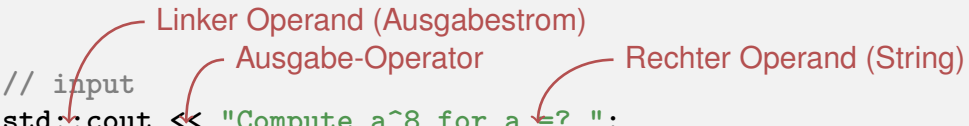


Operatoren und Operanden

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




```
// 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


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

// computati
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)

Operatoren und Operanden


```
// 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