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
- 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.

Zu den Übungen

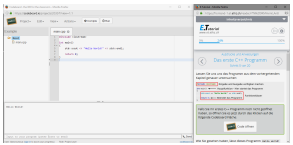
- Bearbeitung der wöchentlichen Uebungsserien ist also freiwillig, wird aber *dringend* empfohlen!

Fehlende Resourcen sind keine Entschuldigung!

Für die Übungen verwenden wir eine Online-Entwicklungsumgebung, benötigt lediglich einen Browser, Internetverbindung und Ihr ETH Login.

Falls Sie keinen Zugang zu einem Computer haben: in der ETH stehen an vielen Orten öffentlich Computer bereit.

Online Tutorial



Zum Einstieg stellen wir ein *Online-C++ Tutorial* zur Verfügung.

Ziel: Ausgleich der unterschiedlichen Programmierkenntnisse.

Schriftlicher Minitest zur *Selbsteinschätzung* in der zweiten Übungsstunde.

Relevantes für die Prüfung

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

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

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

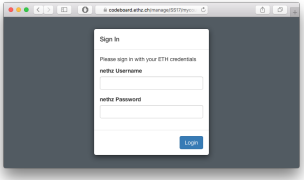
Regel: Sie geben nur eigene Lösungen ab, welche Sie selbst verfasst und verstanden haben.

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.

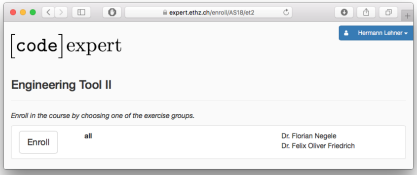
Einschreibung in Übungsgruppen - I

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

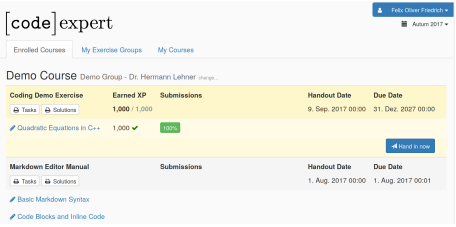


Einschreibung in Übungsgruppen - II

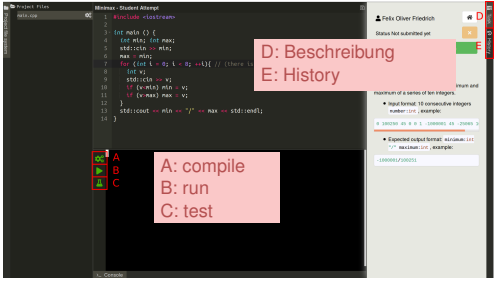
Schreiben Sie sich im folgenden Dialog in eine Übungsgruppe ein.



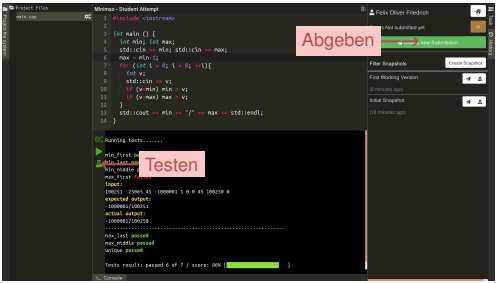
Übersicht



Programmierübung



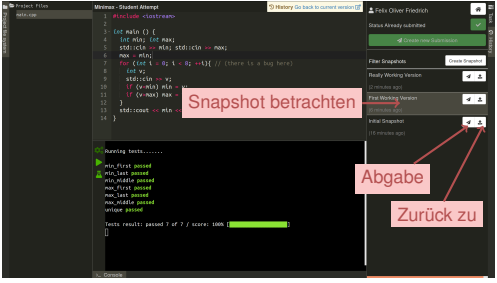
Testen und Abgeben



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



Literatur

- Der Kurs ist ausgelegt darauf, selbsterklärend zu sein.
- Vorlesungsskript gemeinsam mit Informatik für Mathematiker und Physiker. Insbesondere erster Teil.
- Empfehlenswerte Literatur
  - B. Stroustrup. *Einführung in die Programmierung mit C++*, Pearson Studium, 2010.
  - B. Stroustrup, *The C++ Programming Language* (4th Edition) Addison-Wesley, 2013.
  - A. Koenig, B.E. Moo, *Accelerated C++*, Adddison Wesley, 2000.
  - B. Stroustrup, *The design and evolution of C++*, Addison-Wesley, 1994.

Credits

- Vorlesung:
  - Ursprüngliche Fassung von Prof. B. Gärtner und Dr. F. Friedrich
  - Mit Änderungen von Dr. F. Friedrich, Dr. H. Lehner, Dr. M. Schwerhoff
- Skript: Prof. B. Gärtner
- Code Expert: Dr. H. Lehner, David Avanthay und anderen

Andere Quellen werden hier am Rand in dieser Form angegeben.

Informatik

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

Malte Schwerhoff, Felix Friedrich

HS 2018

1. Einführung

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

Was ist Informatik?

- 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)

http://larc.unt.edu/ian/research/cseducation/fellows1991.pdf

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: *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.

TECHNIK

Das programmgesteuerte Rechengerät an der Eidgenössischen Technischen Hochschule in Zürich

Neue Zürcher Zeitung, 30. August 1950

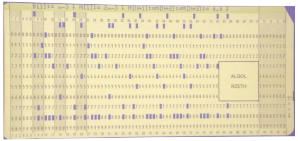
ETH: Pionierin der modernen Informatik

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

http://de.wikipedia.org/wiki/ERMETH

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

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 C++.
- Also:
  nicht nur,
  aber auch Programmierkurs.

Algorithmus: Kernbegriff der Informatik

Algorithmus:

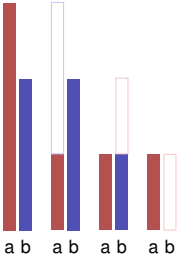
- Handlungsanweisung zur schrittweisen Lösung eines Problems
- Ausführung erfordert keine Intelligenz, nur Genauigkeit (sogar Computer können es)
- nach *Muhammed al-Chwarizmi*, Autor eines arabischen Rechen-Lehrbuchs (um 825)

"Dixit algorizmi..." (lateinische Übersetzung)

http://de.wikipedia.org/wiki/Algorithmus

Der älteste nichttriviale Algorithmus

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



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

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

Ergebnis: a.

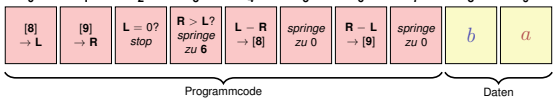
Algorithmen: 3 Abstraktionsstufen

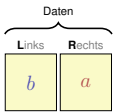
1. **Kernidee** (abstrakt)**:**
   Die Essenz eines Algorithmus' („Heureka-Moment")
2. **Pseudocode** (semi-detailliert)**:**
   Für Menschen gemacht (Bildung, Korrektheit- und Effizienzdiskussionen, Beweise)
3. **Implementierung** (sehr detailliert)**:**
   Für Mensch & Computer gemacht (les- & ausführbar,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

Euklid: Kernidee und Pseudocode gesehen, Implementierung noch nicht

Euklid in der Box

Speicher



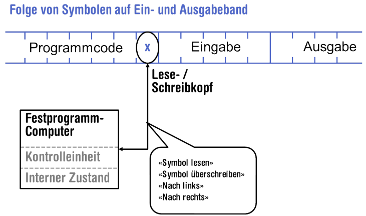


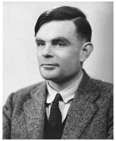
Solange $b \neq 0$
  Wenn $a > b$ dann
    $a \leftarrow a - b$
  Sonst:
    $b \leftarrow b - a$

Ergebnis: a.

Computer – Konzept

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

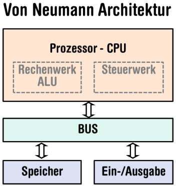



Alan Turing

http://en.wikipedia.org/wiki/Alan_Turing

Computer – Umsetzung

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




Konrad Zuse


John von Neumann

http://www.hs.uni-hamburg.de/DE/GNT/hh/biogr/zuse.htm
http://commons.wikimedia.org/wiki/File:John_von_Neumann.jpg

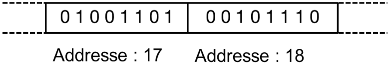
Computer

Zutaten der *Von Neumann Architektur:*

- Hauptspeicher (RAM) für Programme *und* Daten
- Prozessor (CPU) zur Verarbeitung der Programme und Daten
- I/O Komponenten zur Kommunikation mit der Aussenwelt

Speicher für Daten *und* Programm

- Folge von Bits aus $\{0, 1\}$.
- Programmzustand: Werte aller Bits.
- Zusammenfassung von Bits zu Speicherzellen (oft: 8 Bits = 1 Byte).
- Jede Speicherzelle hat eine Adresse.
- Random Access: Zugriffszeit auf Speicherzelle (nahezu) unabhängig von ihrer Adresse.



Prozessor

Der Prozessor (CPU)

- führt Befehle in Maschinensprache aus
- hat eigenen "schnellen" Speicher (Register)
- kann vom Hauptspeicher lesen und in ihn schreiben
- beherrscht eine Menge einfachster Operationen (z.B. Addieren zweier Registerinhalte)

Programmieren

- Mit Hilfe einer *Programmiersprache* wird dem Computer eine Folge von Befehlen erteilt, damit er genau das macht, was wir wollen.
- Die Folge von Befehlen ist das *(Computer)-Programm*.

The Harvard Computers, Menschliche Berufsrechner, ca. 1890

http://en.wikipedia.org/wiki/Harvard_Computers

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 ...
- 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)
- 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!

Programmiersprachen – Einordnung

Unterscheidung in

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

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

Warum C++?

- C++versieht C mit der Mächtigkeit der Abstraktion einer höheren Programmiersprache
- In diesem Kurs: C++ als Hochsprache eingeführt (nicht als besseres C)
- Vorgehen: Traditionell prozedural → objekt-orientiert

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
```


C++: Fehlerarten illustriert an deutscher Sprache

Das Auto fuhr zu schnell.	Syntaktisch und semantisch korrekt.
DasAuto fuh r zu sxhnell.	Syntaxfehler: Wortbildung.
Rot das Auto ist.	Syntaxfehler: Satzstellung.
Man empfiehlt dem Dozenten nicht zu widersprechen	Syntaxfehler: Satzzeichen fehlen .
Sie ist nicht gross und rothaarig.	Syntaktisch korrekt aber mehrdeutig. [kein Analogon]
Die Auto ist rot.	Syntaktisch korrekt, doch semantisch fehlerhaft: Falscher Artikel. [Typfehler]
Das Fahrrad galoppiert schnell.	Syntaktisch und grammatikalisch korrekt! Semantisch fehlerhaft. [Laufzeitfehler]
Manche Tiere riechen gut.	Syntaktisch und semantisch korrekt. Semantisch mehrdeutig. [kein Analogon]

Syntax und Semantik von C++

Syntax:

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

Semantik:

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

Syntax und Semantik von C++

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

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

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)

Sprachbestandteile am Beispiel

- Kommentare/Layout
- Include-Direktiven
- Die main-Funktion
- Werte, Effekte
- Typen, Funktionalität
- Literale
- Variablen
- Konstanten
- Bezeichner, Namen
- Objekte
- **Ausdrücke**
- L- und R-Werte
- Operatoren
- Anweisungen

Das erste C++ Programm Wichtigste Bestandteile...

```
// Program: power8.cpp
// Raise a number to the eighth power.
#include <iostream>
int main() {
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a; ←——— Anweisungen: Mache etwas (lies a ein)!
    // computation
    int b = a * a; // b = a^2 ←——— Ausdrücke: Berechne einen Wert (a²)!
    b = b * b;     // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
```


Verhalten eines Programmes

Zur Compilationszeit:

- vom Compiler akzeptiertes Programm (syntaktisch korrektes C++)
- Compiler-Fehler

Zur Laufzeit:

- korrektes Resultat
- inkorrektes Resultat
- Programmabsturz
- Programm *terminiert* nicht (Endlosschleife)

“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

Kommentare

- hat jedes gute Programm,
- dokumentieren, *was* das Programm *wie* macht und wie man es verwendet und
- werden vom Compiler ignoriert.
- Syntax: „Doppelslash“ `// bis Zeilenende.`

Ignoriert werden vom Compiler ausserdem

- Leerzeilen, Leerzeichen,
- Einrückungen, die die Programmlogik widerspiegeln (sollten)

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;
```

Include-Direktive

Funktionsdeklaration der main-Funktion

Include-Direktiven

C++ besteht aus

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

```
#include <iostream>
```

- macht Ein/Ausgabe verfügbar

Die Hauptfunktion

Die `main`-Funktion

- existiert in jedem C++ Programm
- wird vom Betriebssystem aufgerufen
- wie eine mathematische Funktion ...
  - Argumente (bei `power8.cpp`: keine)
  - Rückgabe**wert** (bei `power8.cpp`: 0)
- ... aber mit zusätzlichem *Effekt*.
  - Lies eine Zahl ein und gib die 8-te Potenz aus.

Anweisungen: Mache etwas!

```
int main() {
    // input
    std::cout << "Compute a^8 for a =? ";
    int a;
    std::cin >> a;
    // computation
    int b = a * a; // b = a^2
    b = b * b;     // b = a^4
    // output b * b, i.e., a^8
    std::cout << a << "^8 = " << b * b << "\n";
    return 0;
}
```

Ausdrucksanweisungen

Rückgabeanweisung

Anweisungen

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

Ausdrucksanweisungen

- haben die Form

  expr;

  wobei *expr* ein Ausdruck ist
- Effekt ist der Effekt von *expr*, der Wert von *expr* wird ignoriert.

Beispiel: `b = b*b;`

Rückgabeanweisungen

- treten nur in Funktionen auf und sind von der Form

  `return` expr;

  wobei *expr* ein Ausdruck ist
- spezifizieren Rückgabewert der Funktion

Beispiel: `return 0;`

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

Werte und Effekte

- bestimmen, was das Programm macht,
- sind rein semantische Konzepte:
  - Zeichen `0` bedeutet Wert $0 \in \mathbb{Z}$
  - `std::cin >> a;` bedeutet Effekt "Einlesen einer Zahl"
- hängen vom Programmzustand (Speicherinhalte / Eingaben) ab

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

Deklarationsanweisungen

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

  Beispiel: `int a;`
- können Variablen auch initialisieren

  Beispiel: `int b = a * a;`

Typen und Funktionalität

`int`:

- C++ Typ für ganze Zahlen,
- entspricht $(\mathbb{Z}, +, \times)$ in der Mathematik

In C++ hat jeder Typ einen Namen sowie

- Wertebereich (z.B. ganze Zahlen)
- Funktionalität (z.B. Addition/Multiplikation)

Fundamentaltypen

C++ enthält fundamentale Typen für

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

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*
- sind im Programmtext „sichtbar“

Beispiel

`int a;` definiert Variable mit

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

Objekte

- repräsentieren Werte im Hauptspeicher
- haben *Typ*, *Adresse* und *Wert* (Speicherinhalt an der Adresse),
- können benannt werden (Variable) ...
- ... aber auch anonym sein.

Anmerkung

Ein Programm hat eine *feste* Anzahl von Variablen. Um eine variable Anzahl von Werten behandeln zu können, braucht es „anonyme“ Adressen, die über temporäre Namen angesprochen werden können (→ Informatik 1).

Bezeichner und Namen

(Variablen-)Namen sind Bezeichner:

- erlaubt: A,...,Z; a,...,z; 0,...,9;_
- erstes Zeichen ist Buchstabe.

Es gibt noch andere Namen:

- `std::cin` (qualifizierter Name)

Ausdrücke: Berechne einen Wert!

- repräsentieren *Berechnungen*
- sind entweder primär (`b`)
- oder zusammengesetzt (`b*b`)...
- ...aus anderen Ausdrücken, mit Hilfe von Operatoren
- haben einen Typ und einen Wert

Analogie: Baukasten

Ausdrücke Baukasten

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;
```

Zweifach zusammengesetzter Ausdruck

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


Ausdrücke (Expressions)

- repräsentieren *Berechnungen*,
- sind *primär* oder *zusammengesetzt* (aus anderen Ausdrücken und Operationen)

`a * a`
zusammengesetzt aus
Variablenname, Operatorsymbol, Variablenname
Variablenname: primärer Ausdruck

- können geklammert werden

`a * a` ist äquivalent zu `(a * a)`

Ausdrücke (Expressions)

haben *Typ*, *Wert* und *Effekt* (potenziell).

Beispiel

`a * a`

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `a` und `a`
- Effekt: keiner.

Beispiel

`b = b * b`

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `b` und `b`
- Effekt: Weise `b` diesen Wert zu.

Typ eines Ausdrucks ist fest, aber Wert und Effekt werden erst durch die *Auswertung* des Ausdrucks bestimmt.

L-Werte und R-Werte

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

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

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

R-Wert

L-Wert (Ausdruck + Adresse)

L-Wert (Ausdruck + Adresse)

R-Wert

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

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

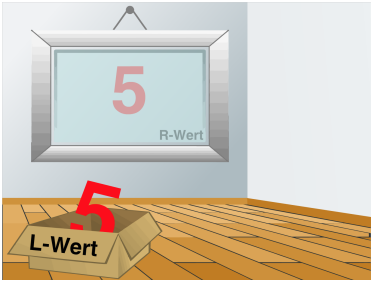
L-Werte und R-Werte

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

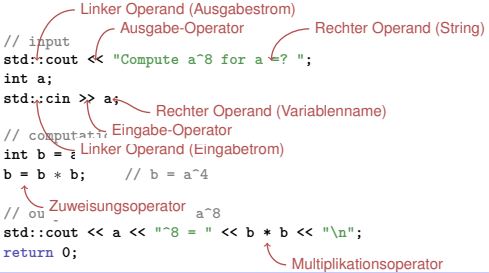
- Ausdruck der kein L-Wert ist

  Beispiel: Literal 0
- 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



Operatoren

Operatoren

- machen aus Ausdrücken (*Operanden*) neue zusammengesetzte Ausdrücke
- spezifizieren für die Operanden und das Ergebnis die Typen, und ob sie L- oder R-Werte sein müssen
- haben eine Stelligkeit

Multiplikationsoperator ∗

- erwartet zwei R-Werte vom gleichen Typ als Operanden (Stelligkeit 2)
- "gibt Produkt als R-Wert des gleichen Typs zurück", das heisst formal:
  - Der zusammengesetzte Ausdruck ist ein R-Wert; sein Wert ist das Produkt der Werte der beiden Operanden

Beispiele: `a * a` und `b * b`

Zuweisungsoperator =

- Linker Operand ist **L**-Wert,
- **R**echter Operand ist **R**-Wert des gleichen Typs.
- Weist linkem Operanden den Wert des rechten Operanden zu und gibt den linken Operanden als L-Wert zurück

Beispiele: `b = b * b` und `a = b`

Vorsicht, Falle!

Der Operator = entspricht dem Zuweisungsoperator in der Mathematik (:=), nicht dem Vergleichsoperator (=).

Eingabeoperator >>

- linker Operand ist L-Wert (*Eingabestrom*)
- rechter Operand ist L-Wert
- weist dem rechten Operanden den nächsten Wert aus der Eingabe zu, *entfernt ihn aus der Eingabe* und gibt den Eingabestrom als L-Wert zurück

Beispiel: `std::cin >> a` (meist Tastatureingabe)

- Eingabestrom wird verändert und muss deshalb ein L-Wert sein!

Ausgabeoperator <<

- linker Operand ist L-Wert (*Ausgabestrom*)
- rechter Operand ist R-Wert
- gibt den Wert des rechten Operanden aus, fügt ihn dem Ausgabestrom hinzu und gibt den Ausgabestrom als L-Wert zurück

Beispiel: `std::cout << a` (meist Bildschirmausgabe)

- Ausgabestrom wird verändert und muss deshalb ein L-Wert sein!

Ausgabeoperator <<

Warum Rückgabe des Ausgabestroms?

- erlaubt Bündelung von Ausgaben

```
std::cout << a << "^8 = " << b * b << "\n"

ist wie folgt logisch geklammert

((((std::cout << a) << "^8 = ") << b * b) << "\n")
```

- `std::cout << a` dient als linker Operand des nächsten << und ist somit ein L-Wert, der kein Variablenname ist.

power8_exact.cpp

- Problem mit `power8.cpp`: grosse Eingaben werden nicht korrekt behandelt
- Grund: Wertebereich des Typs int ist beschränkt (siehe nächste VL)
- Lösung: verwende einen anderen Typ, z.B. `ifm::integer`

power8_exact.cpp

```
// Program: power8_exact.cpp
// Raise a number to the eighth power,
// using integers of arbitrary size

#include <iostream>
#include <IFMP/integer.h>

int main()
{
  // input
  std::cout << "Compute a^8 for a =? ";
  ifmp::integer a;
  std::cin >> a;

  // computation
  ifmp::integer 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;
}
```