Vorkurs Informatik (D-ITET)

Malte Schwerhoff

September 2018

http://lec.inf.ethz.ch/itet/informatik0/2018

Kontext dieser Vorlesung

- **Informatik 0** (Vorkurs): Erste Programmiererfahrungen sammeln
- **Informatik 1**: Theoretische und praktische Grundlagen der Informatik
- **Informatik 2**: Algorithmen und Datenstrukturen
- **Technische Informatik 1**: Logische und physikalische Strukturen digitaler Systeme
- **Technische Informatik 2**: Wichtige Komponenten von Betriebssystemen
- ... und weitere Fächer mit (mehr oder weniger direktem) Informatikbezug

Kursziele

- Erste Programmiererfahrung vermitteln
- An der „Informatik-Oberfläche“ kratzen
- Dieser Kurs kann leider nicht jahrelange Programmiererfahrung (Schule, Hobby) ausgleichen ...
- ... aber er sollte Ihnen den Einstieg in Informatik 1 erleichtern

Kursaufbau: Leistungskontrolle

- Keine Prüfung
- Stattdessen müssen zwei Programmierprojekte bestanden werden

Kursaufbau: Stundenplan

Woche(n)	Programm
1	Vorlesung: Do 20.09., 13:15 – 15:00 C++-Tutorial, 1. Projekt
2	Präsenzstunden: Mo 24.09. HG F1, Di 25.09. HG E7, 17:00 – 19:00 1. Projekt
3	Abgabe 1. Projekt Vorlesung: Mi 03.10., 13:15 – 15:00 Präsenzstunden: Mo 01.10. HG F1, Di 02.10. HG E7, 17:00 – 19:00 2. Projekt
4-7	Präsenzstunden: Di 9./16./23./30.10, HG E7, 17:00 – 19:00 2. Projekt
7	Abgabe 2. Projekt

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://www.cam.ed.ac.uk/research/discussion/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*

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

Informatik: *Grundlagenwissen*

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

Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Handlungsanweisung zur schrittweisen Lösung eines Problems
- Ausführung erfordert keine Intelligenz, nur Genauigkeit (sogar Computer können es)
- Ältester nichttrivialer Algorithmus: Euklidischer Algorithmus, 3. Jh. v. Chr.
- nach *Muhammed al-Chwarizmi*, Autor eines arabischen Rechen-Lehrbuchs (um 825)

"Dixit algorizmi. . . " (lateinische Übersetzung)

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

Binäre Suche: Problem & Idee

Problem: Finde ein Element in einer *sortierten* Liste

Idee: Wörterbuchsuche — mittig aufschlagen, falls nötig links/rechts weitersuchen

Binäre Suche: Beispiel

Beispiel: Finde 7 in der Liste $[1, 3, 4, 7, 9, 11, 12, 17]$

1	3	4	7	9	11	12	17
1	3	4	7	9	11	12	17
1	3	4	7	9	11	12	17
1	3	4	7	9	11	12	17
1	3	4	7	9	11	12	17
1	3	4	7	9	11	12	17

Binäre Suche: Pseudocode

- Eingabe: Liste sortierter Zahlen L, zu findende Zahl n
- Ausgabe: „Ja“ („Nein“) falls n (nicht) in L

Solange Ausgabe A noch unbekannt ist:
  Falls L leer dann: $A \leftarrow$ „Nein“
  Andernfalls:
    Wähle mittige Zahl L_m aus
    Falls $L_m = n$ dann: $A \leftarrow$ „Ja“
    Falls $n < L_m$ dann: $L \leftarrow$ linke Hälfte von L
    Ansonsten: $L \leftarrow$ rechte Hälfte von L

Binäre Suche: C++-Implementierung

```
std::string binary_search(const int* begin, const int* end,
                          const int n) {
  while (true) {
    if (begin >= end) return "no";
    const int* mid = begin + (end - begin) / 2;
    if (*mid == n) return "yes";
    else if (n < *mid) end = mid;
    else begin = mid + 1;
  }
}
```

Keine Panik: Der C++-Code wird hier nur gezeigt um den Unterschied zwischen einer Algorithmusbeschreibung in Pseudocode und einer konkreten Implementierung zu illustrieren. Die genutzten Sprachkonstrukte werden Sie erst in Informatik 1 kennenlernen.

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)

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)
- Programmieren ist *die* Schnittstelle zwischen Ingenieurwissenschaften und Informatik – der interdisziplinäre Grenzbereich wächst zusehends.
- Programmieren macht Spass (und ist nützlich)!

Programmieren

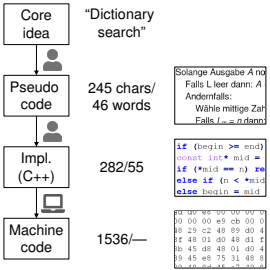
- Mit Hilfe einer *Programmiersprache* wird dem Computer eine Folge von Befehlen erteilt, damit er genau das macht, was wir wollen.
- Die Folge von Befehlen ist das *(Computer)-Programm*.

The Harvard Computers. Menschliche Berufsrechner, ca. 1890

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

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.



Rechengeschwindigkeit

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

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

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

[1] Uniprozessor Computer bei 1GHz

Höhere Programmiersprachen

Wir schreiben Programme (Implementierungen) in einer *höheren Programmiersprache*:

- Kann von Menschen *verstanden* werden kann
- Ist *hardwareunabhängig*
- Beinhaltet *wiederverwendbare* Funktionsbibliotheken

Warum C++?

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

- C++ ist relevant in der Praxis, weit verbreitet und „läuft überall“
- C++ ist standardisiert, d.h. es gibt ein offizielles
- 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.
- 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

2. C++-Sprachkonstrukte am Beispiel

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)

Ein erstes C++-Programm: Vorwort

Die kommenden Folien zeigen ein erstes, interessantes Programm anhand dessen wichtige Bestandteile der Programmiersprache C++ illustriert werden. Die Folien sind eine Art Kurzzusammenfassung des C++-Tutorials; es ist daher ratsam, zuerst das Tutorial durchzuarbeiten.

Das gezeigte Programm ist *nicht* das Programm aus der Vorlesung — letzteres zu reproduzieren ist Teil der ersten Projektaufgabe ☺.

Sprachbestandteile am Beispiel

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

Die Basis: Anweisungen und Ausdrücke

```
// 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;
}
```

Anweisungen: Mache etwas (lies a ein)!

Ausdrücke: Berechne einen Wert (a^2)!

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 (und Linker, Laufzeit)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:

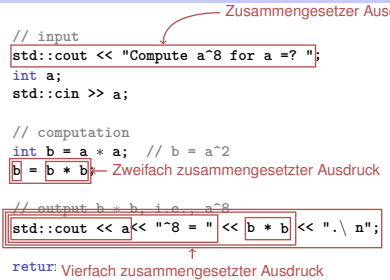
- `std::cin` (qualifizierter Name)

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



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.

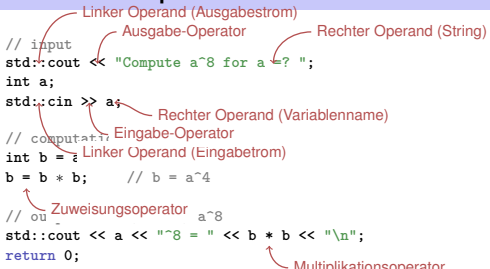
Beispiel

`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.

Operatoren und Operanden Baukasten



Operatoren

Operatoren

- machen aus Ausdrücken (*Operanden*) neue zusammengesetzte Ausdrücke
- spezifizieren für die Operanden und das Ergebnis die Typen
- haben eine Stelligkeit (Anzahl Operanden)

Multiplikationsoperator ∗

- Erwartet zwei Werte vom gleichen Typ als Operanden (Stelligkeit 2)
- Gibt Produkt als Wert des gleichen Typs zurück
- D.h. der zusammengesetzte Ausdruck repräsentiert ist ein Wert; sein Wert ist das Produkt der Werte der beiden Operanden

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

Zuweisungsoperator =

Weist linkem Operanden (i.d.R. eine Variable) den Wert des rechten Operanden zu

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

Vorsicht, Falle!

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

Eingabeoperator >>

- linker Operand ist der *Eingabestrom*
- weist dem rechten Operanden (i.d.R. eine Variable) den nächsten Wert aus der Eingabe zu, *entfernt ihn aus der Eingabe* und gibt den Eingabestrom zurück

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

Ausgabeoperator <<

- linker Operand ist der *Ausgabestrom*
- gibt den Wert des rechten Operanden aus, fügt ihn dem Ausgabestrom hinzu und gibt den Ausgabestrom zurück

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

Ausgabeoperator <<

Warum Rückgabe des Ausgabestroms?

- erlaubt Bündelung von Ausgaben

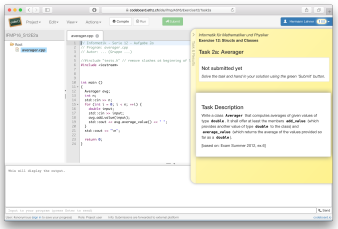
```
std::cout << a << "^8 = " << b * b << "\n"

ist wie folgt logisch geklammert

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

- `std::cout << a` dient als linker Operand (d.h. als Ausgabestrom) des nächsten <<

3. Organisation der Programmierprojekte

Codeboard

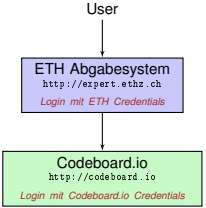
Codeboard ist eine Online-IDE: Programmieren im Browser!



Code Expert

Unser Übungssystem besteht aus zwei unabhängigen Systemen, die miteinander kommunizieren:

- **Das ETH Abgabesystem:** Ermöglicht es uns, Ihre Aufgaben zu bewerten
- **Die Online IDE:** Die Programmierumgebung



Projekte = Übungen

- Code Expert ist für einen „normalen“ Übungsbetrieb ausgelegt, daher die Terminologie „Übung“, „Übungsgruppe“, etc.
- Auf Code Expert sind unsere Programmierprojekte daher Übungen und es gibt genau eine Übungsgruppe

Einschreibung

Codeboard.io Registrierung

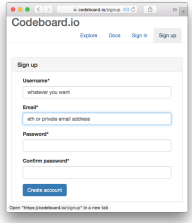
Gehen Sie auf `http://codeboard.io` und erstellen Sie dort ein Konto, bleiben Sie am besten eingeloggt.

Einschreibung auf

Gehen Sie auf `http://expert.ethz.ch/inf0itet18` und schreiben Sie sich in die Übungsgruppe `Students` ein.

Codeboard.io Registrierung

Falls Sie noch keinen **Codeboard.io** Account haben ...



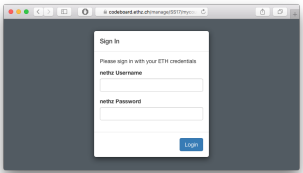
- Wir verwenden die Online IDE **Codeboard.io**
- Erstellen Sie dort einen Account, um Ihren Fortschritt abzuspeichern und später Submissions anzuschauen
- Anmeldedaten können beliebig gewählt werden! *Verwenden Sie nicht das ETH Passwort.*

Codeboard.io Login

Falls Sie schon einen Account haben, loggen Sie sich ein:

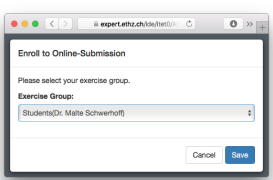
Einschreibung in Übungsgruppen - I

- Besuchen Sie `http://expert.ethz.ch/inf0itet18`
- Loggen Sie sich mit Ihrem nethz-Account ein.



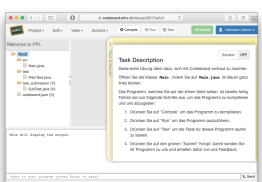
Einschreibung in Übungsgruppen - II

Schreiben Sie sich in diesem Dialog in die Übungsgruppe `Students` ein.



Die erste Übung

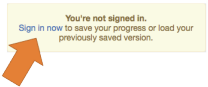
Sie sind nun eingeschrieben und die erste Übung ist geladen.
Folgen Sie den Anweisungen in der gelben Box.



(Screenshot nicht aktuell)

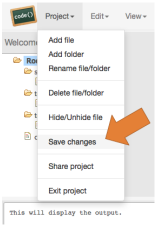
Achtung: Codeboard.io-Login

Achtung! Falls Sie diese Nachricht sehen, klicken Sie auf Sign in now und melden Sie sich dort mit Ihrem **Codeboard.io**-Account ein.



Achtung: Fortschritt speichern

Achtung! Speichern Sie Ihren Fortschritt regelmässig ab. So können Sie jederzeit an einem anderen Ort weiterarbeiten.



Akademische Lauterkeit

Es gilt die Leistungskontrollenverordnung der ETH Zürich

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

Wir prüfen das (zum Teil automatisiert) nach und behalten uns disziplinarische Massnahmen vor.

So ist's OK (siehe Algorithmen-Folien):

- Kernidee: gerne in der Gruppe diskutieren
- Pseudocode: gerne in der Gruppe diskutieren
- Implementierung: *Einzelarbeit!*