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 $\neq$ Computer Science

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

Computer Science $\subseteq$ Informatik

- Die Informatik beschäftigt sich heute auch mit dem Entwurf von schnellen Computern und Netzwerken. . .
- . . . aber nicht als Selbstzweck, sondern zur effizienteren **systematischen Verarbeitung von Informationen**.

Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Rezept zur **systematischen Verarbeitung von Informationen**
- nach *Muhammed al-Chwarizmi*, Autor eines arabischen Rechen-Lehrbuchs (um 825)

“Dixit algorizmi. . . ” (lateinische Übersetzung)

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

Der älteste überlieferte Algorithmus…

…ist der **Euklidische Algorithmus** (aus Euklids *Elementen*, 3. Jh. v. Chr.)

- Eingabe: zwei natürliche Zahlen $a > b$
- Ausgabe: grösster gemeinsamer Teiler $\mathrm{ggT}(a, b)$

Algorithmus am Beispiel:

a	b	$a \operatorname{div} b$	$a \mod b$	
1071	1029	1	42	
1029	42	24	21	
42	21	2	0	$\Rightarrow \mathrm{ggT} = 21$

Informatik $\neq$ EDV-Kenntnisse

EDV-Kenntnisse: *Anwenderwissen*

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

Informatik: *Grundlagenwissen*

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

Informatik in die Schulen!

Imagine the dramatic change which could be possible in just a few years, once we remove the roadblock of the existing ICT curriculum. Instead of children bored out of their minds being taught how to use Word and Excel by bored teachers, we could have 11 year-olds able to write simple 2D computer animations using an MIT tool called Scratch.

Michael Gove, Bildungsminister UK, 2010–2014

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.

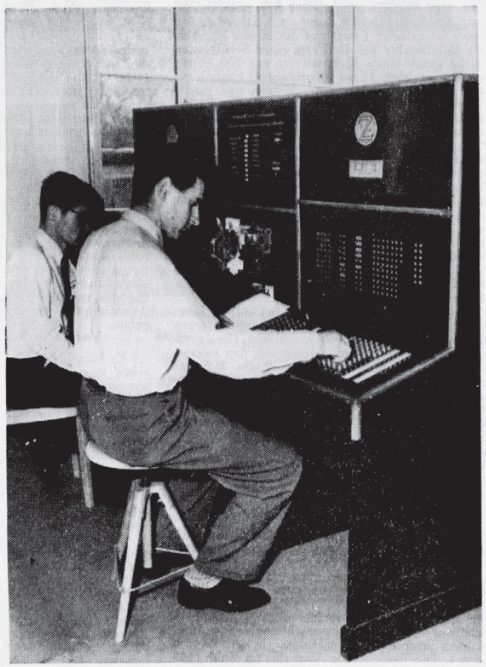


Abb. 2. Der Schaltpult bei der Fertigung eines Rechenplanes. Die Abtaster für den Lochstreifen sind deutlich sichtbar.

Befehle können „bedingt" gegeben werden, d. h. ihre Ausführung wird von der Natur eines errechneten Resultates abhängig gemacht. Erst da-

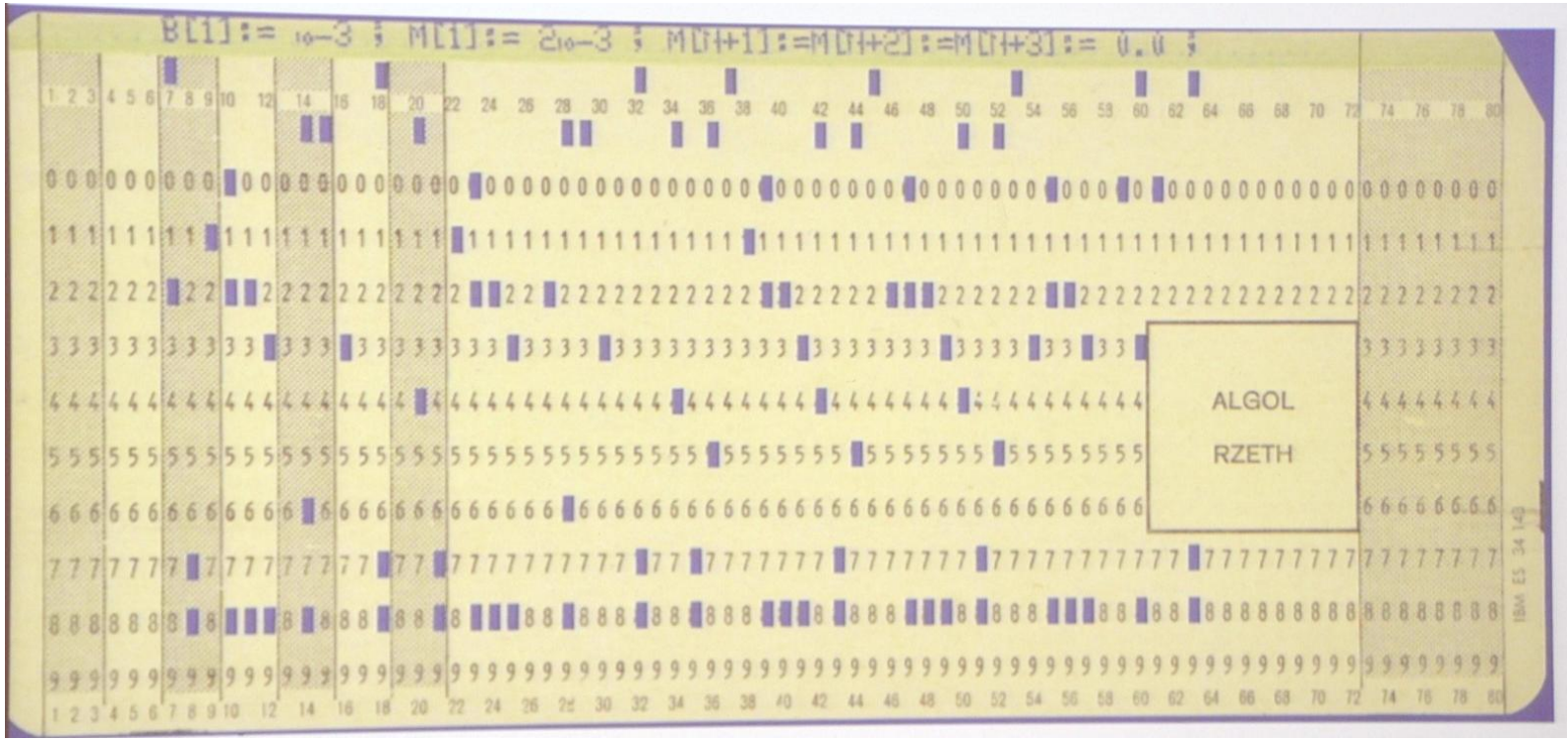
ETH: Pionierin der modernen Informatik

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 definierte Programmiersprache), unter anderem durch Heinz Rutishauer, ETH



ETH: Pionierin der modernen Informatik

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

Die Klasse 1964 heute (mit einigen Gästen)

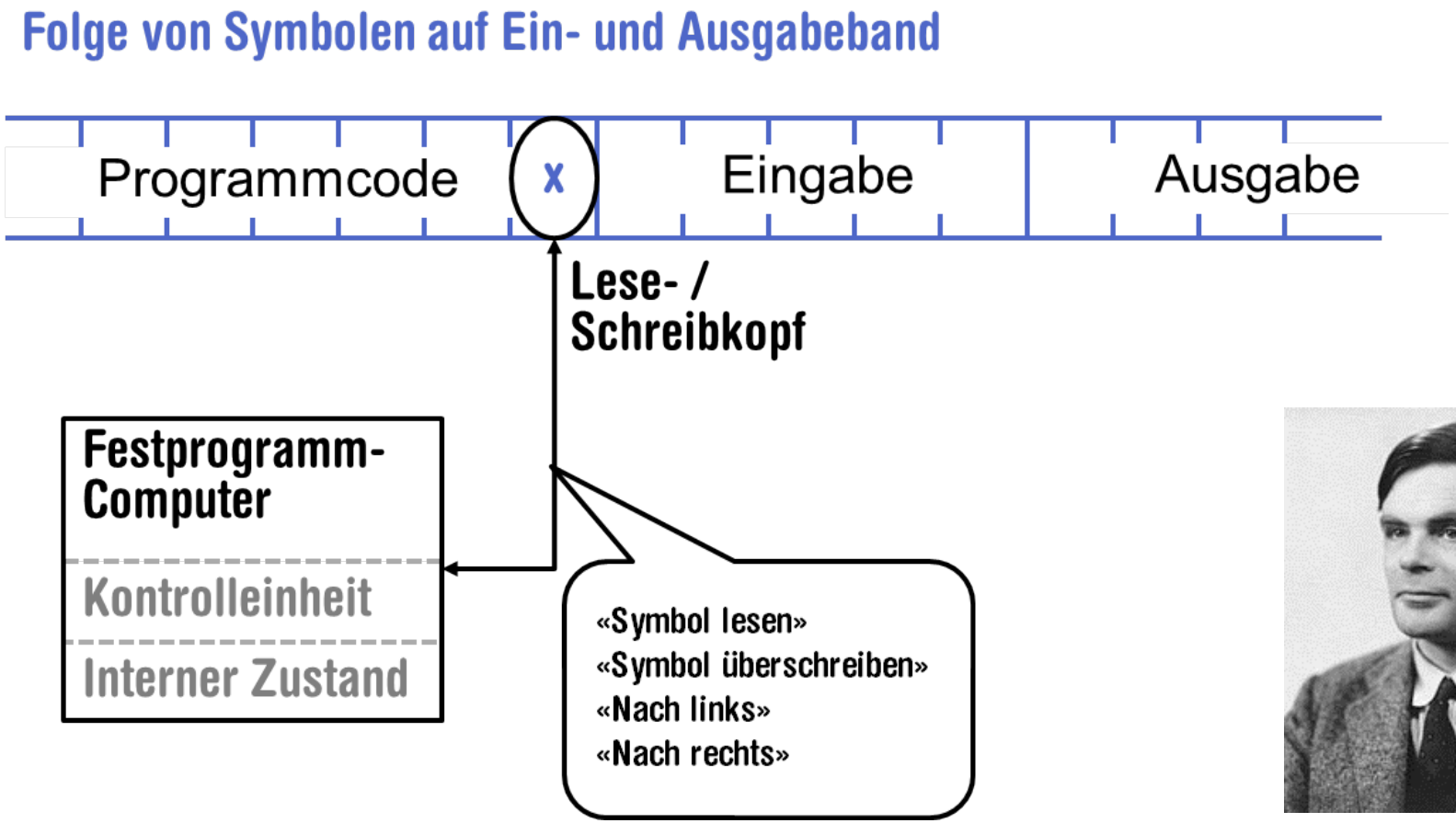
http://www.inf.ethz.ch/news-and-events/spotlights/1964.html

ETH: Pionierin der modernen Informatik

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

Computer – Konzept

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

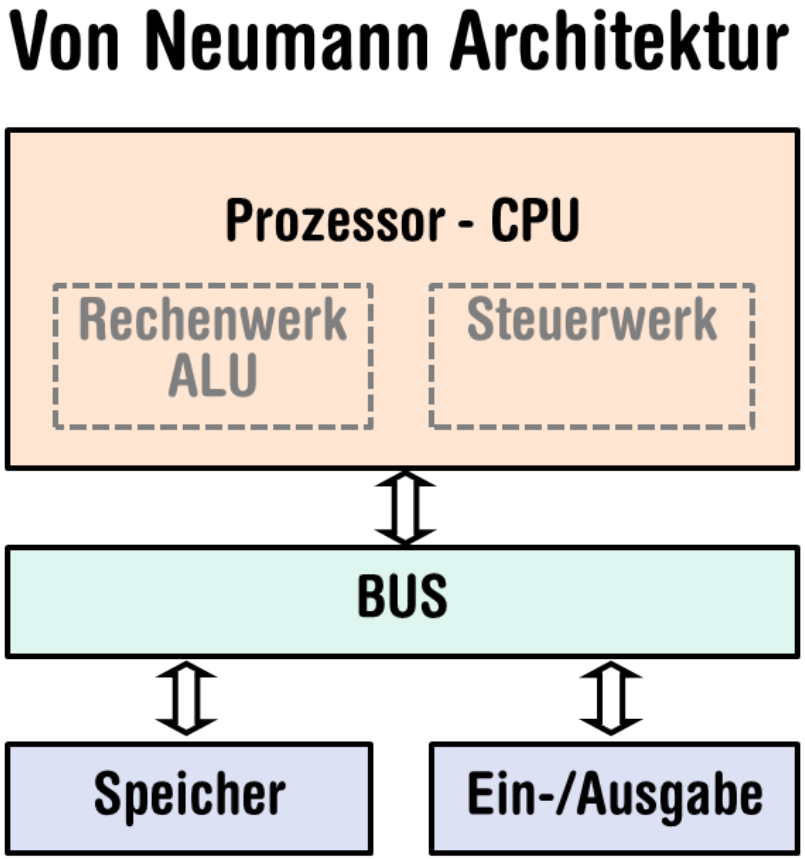


Alan Turing

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

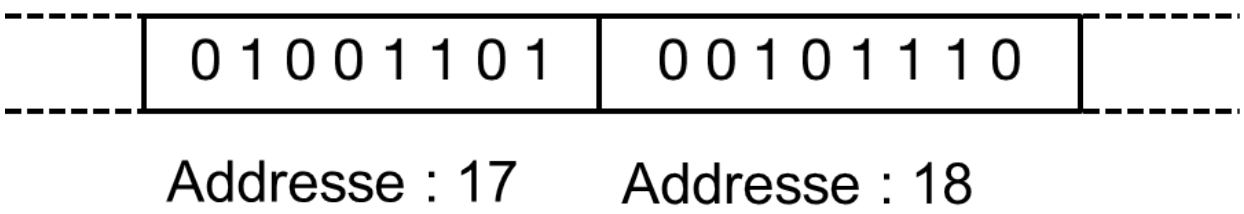
Computer

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

- 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)

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

Inhalt dieser Vorlesung

- Systematisches Problemlösen mit Algorithmen und der Programmiersprache C++.
- Niklaus Wirth laut Internet: *C++ is an insult to the human brain.*
- Ich habe ihn gefragt: Er hat das nie gesagt!
- Vorlesung: *nicht nur,* *aber auch* Programmierkurs.

Deklaratives Wissen

Wissen über *Sachverhalte* – formulierbar in Aussagesätzen.

- Es gibt unendlich viele ganze Zahlen.
- Der Computerspeicher ist endlich.
- x ist eine Wurzel von y, wenn $y = x^2$.

Prozedurales Wissen

Wissen über *Abläufe* – formulierbar in Form von Verarbeitungsanweisungen (kurz: Befehle).

Beispiel: *Algorithmus*[2] zur Approximation von $\sqrt{y}$:

1. Starte mit einem Schätzwert s von $\sqrt{y}$.
2. Ist s^2 nahe genug bei y, dann ist $x := s$ eine gute Approximation der Wurzel von y.
3. Andernfalls erzeuge eine neue Schätzung durch

$$s_{neu} := \frac{s + y/s}{2}.$$

4. Ersetze s durch s_{neu} und gehe zurück zu Schritt 2.

[2] Newton-Methode

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

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 ...
- ...

Darum Programmieren!

- Jedes Verständnis moderner Technologie erfordert Wissen über die grundlegende Funktionsweise eines Computers.
- Programmieren (mit dem Werkzeug Computer) wird zu einer Kulturtechnik wie Lesen und Schreiben (mit den Werkzeugen Papier und Bleistift)
- Die meisten qualifizierten Jobs benötigen zumindest elementare Programmierkenntnisse.

- Programmieren macht Spass!

Programmiersprachen

- Sprache, die der Computer "versteht", ist sehr primitiv (Maschinensprache).
- Einfache Operationen müssen in viele Einzelschritte aufgeteilt werden.
- Sprache variiert von Computer zu Computer.

Höhere Programmiersprache: 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#, Pascal, Modula, Oberon, Java
    vs.
    Python, Tcl, 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#, Objective-C, Modula, Oberon, ...

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

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 $\rightarrow$ Typ gebunden, objekt-orientiert

Deutsch vs. C++

Deutsch

Es ist nicht genug zu wissen,
man muss auch anwenden.
(Johann Wolfgang von Goethe)

C++

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

Syntax und Semantik

- Programme müssen, wie unsere Sprache, nach gewissen Regeln geformt werden.
  - Syntax: Zusammenfügungsregeln für elementare Zeichen (Buchstaben).
  - Semantik: Interpretationsregeln für zusammengefügte Zeichen.
- Entsprechende Regeln für ein Computerprogramm sind einfacher, aber auch strenger, denn Computer sind vergleichsweise dumm.

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 gallopiert 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

- Was *ist* ein C++ Programm?
- Ist es *grammatikalisch* korrekt?

Semantik

- Was *bedeutet* ein Programm?
- Welchen Algorithmus realisiert ein Programm?

Syntax und Semantik von C++

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

- ist das “Gesetz” von C++
- legt Grammatik und Bedeutung von C++-Programmen fest
- enthält seit 2011 Neuerungen für *fortgeschrittenes* Programmieren...
- ...weshalb wir auf diese Neuerungen hier auch nicht weiter eingehen werden.

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 Organsiation all dieser Abläufe (Dateiverwaltung, Editor-, Compiler- und Programmaufruf)

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)

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;
}
```

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

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!

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.

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

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

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 (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 = (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

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,
weitere Beispiele etwas später....

L-Werte und R-Werte

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

- Ausdruck ohne Adresse
- Jeder L-Wert kann als R-Wert benutzt werden (aber nicht umgekehrt).
- Ausdruck der kein L-Wert ist

  Beispiel: Literal `0`
- R-Wert kann seinen Wert *nicht ändern*.

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,
- rechter 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.

Anweisungen (statements)

- 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;`

Deklarationsanweisungen

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

Beispiel: `int a;`

- können Variablen auch initialisieren

Beispiel: `int b = a * a;`

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;`**

`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;
}
```