

# Informatik I

## Grundlagen der systematischen Programmierung

Peter Thiemann

Universität Freiburg, Germany

WS 2008/09

# Organisatorisches

- ▶ **Vorlesung** Di und Do, 11-13 Uhr, HS 101-00-036

- ▶ **Dozent** Prof. Dr. Peter Thiemann

Gebäude 079, Raum 00-015

Telefon: 0761 203 8051/8247

E-mail: `thiemann at informatik uni-freiburg de`

Web: `http://www.informatik.uni-freiburg.de/~thiemann`

- ▶ **Informationen** Homepage der Vorlesung über  
`http://proglang.informatik.uni-freiburg.de/`

# Ankündigungen

## **Einführung in das Studium der Informatik**

Studiendekan Prof. Dr. G. Lausen

Mittwoch, 22.10.2008, 11-13 Uhr

Raum 101-00-036

# Ankündigungen

## **Systemdesignprojekt** (Prof. Dr. L. Reindl)

- ▶ System entwerfen (selbständig agierender Roboter/Fahrzeug)
- ▶ Hardware planen und aufbauen
- ▶ Software programmieren
- ▶ System testen
- ▶ System bei einem Wettbewerb vorführen  
(Ende Januar, Anfang Februar)
  
- ▶ In 4er Gruppen
- ▶ Freiwillig, für Studenten mit Vorkenntnissen
- ▶ Vorbesprechung und Anmeldung:  
Di, 28.10.2008, 9:00, Kinohörsaal 082-00-006

# Literatur

- ▶ Herbert Klaeren, Michael Sperber. *Die Macht der Abstraktion*. Teubner Verlag, 2007.
- ▶ Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi. *How to Design Programs An Introduction to Computing and Programming*. The MIT Press, Cambridge, Massachusetts, London, England, 2001. Siehe auch <http://www.htdp.org/>
- ▶ Max Hailperin, Barbara Kaiser and Karl Knight. *Concrete Abstractions: An Introduction to Computer Science Using Scheme*. Siehe <http://gustavus.edu/+max/concrete-abstractions.html>
- ▶ Harold Abelson, Gerald Jay Sussman with Julie Sussman. *Structure and Interpretation of Computer Programs*, 2. Auflage. MIT Press, 1996.
- ▶ Gerhard Goos. *Vorlesungen über Informatik*, Band 1. Springer-Verlag, 3. Aufl., 2001.

Informatik = Information + Mathematik

*computer science, computing science*

Ingenieurwissenschaft, Systemwissenschaft

# Definition der ACM (Association of Computing Machinery)

Computer science is the systematic study of algorithms and data structures, specifically

1. their formal properties,
2. their mechanical and linguistic realizations, and
3. their applications.

# Core Subjects

- ▶ Principles of Computer Organization
- ▶ Algorithms
- ▶ Theory of Computation
- ▶ Principles of Programming Languages

Vorl. Systeme  
Informatik II  
Informatik III  
Informatik I

# Principles of Programming Languages

- ▶ Definition von Programmiersprachen
  - ▶ Syntax
  - ▶ Semantik
- ▶ Programmiertechniken
- ▶ Abstraktionsmittel
- ▶ Programmierhilfen
- ▶ Analyse von Programmen
- ▶ Implementierung von Programmiersprachen

# Algorithmus und Programm

- ▶ Was kann ein Computer?  
Berechnungsprozesse durchführen

# Algorithmus und Programm

- ▶ Was kann ein Computer?  
Berechnungsprozesse durchführen
- ▶ Was ist ein Berechnungsprozess?  
Transformation von Eingabedaten in Ausgabedaten unter  
Erzeugung von Effekten



# Algorithmus und Programm

- ▶ Was kann ein Computer?  
Berechnungsprozesse durchführen
- ▶ Was ist ein Berechnungsprozess?  
Transformation von Eingabedaten in Ausgabedaten unter  
Erzeugung von Effekten
- ▶ Was ist ein Algorithmus?  
Beschreibung eines Berechnungsprozesses

# Algorithmus und Programm

- ▶ Was kann ein Computer?  
Berechnungsprozesse durchführen
- ▶ Was ist ein Berechnungsprozess?  
Transformation von Eingabedaten in Ausgabedaten unter  
Erzeugung von Effekten
- ▶ Was ist ein Algorithmus?  
Beschreibung eines Berechnungsprozesses
- ▶ Was ist ein Programm?  
Algorithmus ausgedrückt (implementiert) in einer  
Programmiersprache

# Algorithmus und Programm

- ▶ Was kann ein Computer?  
Berechnungsprozesse durchführen
- ▶ Was ist ein Berechnungsprozess?  
Transformation von Eingabedaten in Ausgabedaten unter Erzeugung von Effekten
- ▶ Was ist ein Algorithmus?  
Beschreibung eines Berechnungsprozesses
- ▶ Was ist ein Programm?  
Algorithmus ausgedrückt (implementiert) in einer Programmiersprache
- ▶ Was ist eine Programmiersprache?  
Formale Sprache zum Aufschreiben von Algorithmen  
Formal = künstlich, mit strikten Regeln

# Definition: Algorithmus

Vorschrift zur Durchführung eines Prozesses mit folgenden Eigenschaften:

## Effektivität

Jeder Teilschritt ist ausführbar.

## Determiniertheit

Der nächstfolgende Teilschritt ist immer festgelegt.

## Fintheit

Die Vorschrift ist endlich.

## Terminierung

Der Prozess endet nach endlich vielen Teilschritten.

## Generalität

Die Vorschrift kann eine Klasse von Problemen lösen.

## Präzision

Die Bedeutung jedes Schritts ist eindeutig festgelegt.

## Beispiel: (Alltags-) Algorithmus

„Fülle einen Einkaufswagen gemäss Einkaufszettel“

1. Wähle eine Ware vom Einkaufszettel aus.
2. Fahre mit dem Einkaufswagen zum Standort der Ware.
3. Lade die gewünschte Menge in den Einkaufswagen.
4. Streiche die Ware vom Einkaufszettel.
5. Falls noch Waren auf dem Einkaufszettel, fahre mit 1 fort.

# Weitere Beispiele für Algorithmen

- ▶ Schriftliche Multiplikation
- ▶ Kochrezepte
- ▶ Wegbeschreibungen
- ▶ Spielregeln

# Aufschreiben von Algorithmen

## Programmiersprachen

- ▶ Systemprogrammiersprachen
  - ▶ Nah am Berechnungsmodell der Maschine
  - ▶ Abbildung auf Maschine offensichtlich
- ▶ Höhere Programmiersprachen
  - ▶ Idealisiertes Berechnungsmodell
  - ▶ Abbildung auf Maschine einfach
- ▶ Deklarative Programmiersprachen
  - ▶ Einfacheres Berechnungsmodell
  - ▶ Abbildung auf Maschine schwierig

# Elemente von Programmiersprachen

## ▶ Grundbausteine

- ▶ Schreibweisen für Konstanten (Literele)  
*vgl. Wörter mit fester Bedeutung (aus dem Wörterbuch)*
- ▶ Namen (Bezeichner, Identifier)  
*Wörter mit frei wählbarer Bedeutung*

## ▶ Kombinationsmittel

- ▶ zur Konstruktion von Programmstücken aus vorhandenen Programmstücken
- ▶ *vgl. Grammatik*

## ▶ Abstraktionsmittel

- ▶ Benennung von Programmstücken

# Der Programmierprozess

1. Konstruieren von Programmstücken mit Hilfe von Grundbausteinen und Kombinationsmitteln.
2. Benennen und Beschreiben von Programmstücken mit Hilfe von Abstraktionsmitteln.
3. Die so abstrahierten Programmstücke vergessen und nur die Namen weiter verwenden.
4. Weiter bei 1 bis Problem gelöst.

# Der Programmierprozess

1. Konstruieren von Programmstücken mit Hilfe von Grundbausteinen und Kombinationsmitteln.
2. Benennen und Beschreiben von Programmstücken mit Hilfe von Abstraktionsmitteln.
3. Die so abstrahierten Programmstücke vergessen und nur die Namen weiter verwenden.
4. Weiter bei 1 bis Problem gelöst.
5. Oft auch Trick „Top-down design“:
  - ▶ Betrachte ein Teilproblem als gelöst
  - ▶ Benenne und beschreibe es
  - ▶ Verschiebe seine Lösung auf später

# Was jedes Programm braucht

- ▶ Datenanalyse
- ▶ Vertrag
  - ▶ Beschreibung der Eingabe- und Ausgabe-Daten
  - ▶ Beschreibung der Wirkung des Programms
- ▶ Testumgebung (Beispiele)
- ▶ Definitionen (Programmstücke)

## Beispiel: Parkplatzproblem

Auf einem Parkplatz stehen PKWs und Motorräder. Es handelt sich um  $n$  Fahrzeuge mit  $r$  Rädern.  
Bestimme die Anzahl  $P$  der PKWs.

# Beispiel: Parkplatzproblem

Auf einem Parkplatz stehen PKWs und Motorräder. Es handelt sich um  $n$  Fahrzeuge mit  $r$  Rädern.

Bestimme die Anzahl  $P$  der PKWs.

## Lösungsansatz: Lineares Gleichungssystem

Sei  $M$  die Anzahl der Motorräder.

$$\begin{aligned}M + P &= n \\2M + 4P &= r\end{aligned}$$

# Lösung des Gleichungssystems

Multipliziere erste Gleichung mit  $-2$  und addiere beide Gleichungen.

$$\begin{array}{rclcl} -2M & + & -2P & = & -2n \\ 2M & + & 4P & = & r \\ \hline & & 2P & = & r - 2n \end{array}$$

Also:  $P = r/2 - n$

# Probleme mit der Lösung

$$P(n, r) = r/2 - n$$

▶  $P(3, 9) = 1.5$

# Probleme mit der Lösung

$$P(n, r) = r/2 - n$$

►  $P(3, 9) = 1.5$



# Probleme mit der Lösung

$$P(n, r) = r/2 - n$$

- ▶  $P(3, 9) = 1.5$

- ▶  $P(5, 2) = -4$

# Probleme mit der Lösung

$$P(n, r) = r/2 - n$$

▶  $P(3, 9) = 1.5$

▶  $P(5, 2) = -4$

▶  $P(2, 10) = 3$

# Vollständige Spezifikation (Parkplatzproblem)

Auf einem Parkplatz stehen PKWs und Motorräder. Es handelt sich um  $n$  Fahrzeuge mit  $r$  Rädern.

Bestimme die Anzahl  $P$  der PKWs.

**Eingabe:**  $n, r \in \mathbf{N}$

**Vorbedingung:**  $r$  ist gerade,  $2n \leq r \leq 4n$

**Ausgabe:**  $P \in \mathbf{N}$ , falls die Nachbedingung erfüllbar, sonst „Keine Lösung“

**Nachbedingung:** Für gewisse  $M, P \in \mathbf{N}$  gilt

$$\begin{array}{rclcl} M & + & P & = & n \\ 2M & + & 4P & = & r \end{array}$$

# Fazit

- ▶ Ein Programm beschreibt einen Berechnungsprozess zur Lösung eines Problems.
- ▶ Eine Spezifikation/Vertrag ist ein wichtiger Schritt zur Lösung eines Problems.
- ▶ Eingaben außerhalb des Vertrags erzeugen unsinnige Ausgaben.
- ▶ Programme brauchen Verträge!