

Algorithmisches Denken und imperative Programmierung

Janis Voigtländer

Universität Bonn

Wintersemester 2011/12

Wir

- ▶ Ihr Dozent: Jun.-Prof. Dr. Janis Voigtländer
Institut für Informatik, Abteilung III
<http://www.iai.uni-bonn.de/~jv/>

Arbeitsgebiet: Programmiersprachen

- ▶ Assistent: Dipl.-Math. Daniel Seidel
Institut für Informatik, Abteilung III
<http://preview.tinyurl.com/seideld>
- ▶ Tutoren: Oussama Abdallah
Matthias Bartsch
Marcel Brandt
Moritz Fürneisen
Andy Jäger

Sie

1. erfreulich viele Erstsemestler im Bachelorstudiengang Informatik
2. die ersten Studierenden im neuen Lehramtstudiengang Informatik
3. Studierende mit Nebenfach Informatik

Zu 1., lediglich wenn Programmierung I+II statt Programmierung II+III als Pflichtmodule gewählt.

1.	Logik u. Diskrete Strukturen 9 LP	Technische Informatik 9 LP	Informationssysteme 6 LP	Imperative Programmierung 6 LP	Techniken des wissenschaftlichen Arbeitens 4 LP	28/ 30
2.	Analysis 9 LP	Lineare Algebra 9 LP	Systemnahe Informatik 6 LP	Objektorientierte Softwareentwicklung 6 LP		30
3.	Angewandte Mathematik 6 LP	Algorithmen u. Berechnungskomplexität I 9 LP	Software-technologie 9 LP	Systemnahe Programmierung 6 LP		30/ 28
4.		Algorithmen u. Berechnungskomplexität II 6 LP				6
5.						
6.						

Grundlagen der
Praktischen Informatik
(42/94 LP)

alternativ:
Imperative **oder**
Systemnahe
Programmierung

Diese Veranstaltung

- ▶ Vorlesungstermine:
 - ▶ 2 SWS
 - ▶ montags 12:30–14:00 (Hörsaal 1+2)
 - ▶ 17.10.11–30.01.12, 14 Termine
- ▶ Übungstermine:
 - ▶ 2 SWS
 - ▶ Termine über die Woche verteilt
 - ▶ Einschreibung/Zuordnung per URS sollte erfolgt sein!
 - ▶ ab 25.10.11
 - ▶ Ausfall: 01.11.11 (Feiertag), 07.12.11 (Dies Academicus)
- ▶ „Aufwand“: 6 LP = 180 Stunden Workload!
- ▶ weitere Infos (zu Terminen und anderem) auf Webseite:
 - ▶ zunächst <http://www.iai.uni-bonn.de/~jv/teaching/adip/>
 - ▶ später eCampus/ILIAS

eCampus - Magazin - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

eCampus - Magazin

uni-bonn.de <https://ilias.uni-bonn.de/ilias/> Google

Sie sind nicht angemeldet. Anmelden Sprache | BASIS | Blackboard

Suche

universität bonn eCampus
Lehr- und Lernplattform

Magazin Suche Zuletzt besucht

Magazin

Magazin

Kategorien

- Studiengänge ▼ Aktionen
Hier finden Sie alle Kurse zu den Lehrveranstaltungen aus dem elektronischen Vorlesungsverzeichnis (BASIS).
Um dieses Objekt zu nutzen, müssen Sie angemeldet sein und entsprechende Zugriffsrechte besitzen.
- Fakultäten ▼ Aktionen
Weitere E-Learning-Angebote der Fakultäten, Fachbereiche, Institute und Abteilungen
Um dieses Objekt zu nutzen, müssen Sie angemeldet sein und entsprechende Zugriffsrechte besitzen.
- Zentrale Einrichtungen ▼ Aktionen

Aktuelles

Neue Schulungstermine

Ab September bieten wir wieder eine Reihe von Schulungen zum Thema »Kurse einrichten mit dem neuen eCampus« an.

[» Zur Anmeldung \(Login erforderlich\)](#)

Jetzt neu: eCampus-Handbuch für Lehrende

Ab sofort finden Sie unter »Support« das neue eCampus-Handbuch für Lehrende.

Weitere Details

Zur Übungsdurchführung:

- ▶ Übungen teils in normalen, teils in Kleinstgruppen
- ▶ Einteilung in Kleinstgruppen beim ersten Termin
- ▶ erstes Übungsblatt morgen online
- ▶ zu bearbeiten jeweils bis Sonntag abend
- ▶ wichtig: Informatik-Rechnerzugang

Zur Klausur:

- ▶ voraussichtlich am 15.02.12, 12–14 Uhr
- ▶ Zulassungsbedingungen:
 - ▶ regelmäßige Übungsteilnahme (maximal zwei Fehlzeiten)
 - ▶ 50% der Übungspunkte
 - ▶ Achtung: elektronische Abgaben, harte Deadlines
- ▶ Nachklausur voraussichtlich am 21.03.12

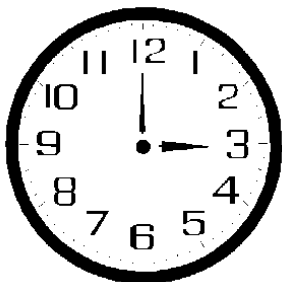
Weiteres zur Organisation

Auf der Webseite finden Sie:

- ▶ Vorlesungsfolien
- ▶ Übungsaufgaben
- ▶ Termine
- ▶ wichtige Mitteilungen

To Do (vom „0. Übungsblatt“):

- ▶ ggfs. noch Übungsnachmeldung
- ▶ Informatik-Rechnerzugang
- ▶ Vertrautmachen mit Compiler
- ▶ Erreichbarkeit per Email unter HRZ-Account sicherstellen!
- ▶ Freischaltung/Anmeldung BASIS



Meldung zur Bachelorprüfung als Ganzes beim Prüfungsausschuss Informatik:

Montag, 24.10.2011, 11:00 – 12:30 Uhr

und

Dienstag, 25.10.2011, 12:00 – 13:30 Uhr

bei Frau König im Foyer in der Römerstraße

mitzubringen:

- Studentenausweis
- Personalausweis
- Original und Kopie d. Abiturzeugnisses
- ausgefülltes Meldeformular
(ebenfalls im Web, unter derselben URL wie BaPO)



2100	Konto: Pflichtbereich		1,1	110,0	
612100011	Modul INF011: Logik und diskrete Strukturen	WS 07/08	1,3 bestanden	8,0	
612200011	Prüfung INF011: Logik und diskrete Strukturen	WS 07/08	18.02.2008	1,3 bestanden	8,0
612100012	Modul INF012:	WS 07/08	1,0 bestanden	6,0	

Zu jeder Modulprüfung:

- verbindliche elektronische **Anmeldung** über BASIS-POS im Internet
- keine Zulassung zur Prüfung ohne Anmeldung
- **Abmeldung** bis eine Woche vor Prüfungstermin möglich
- POS-Zugang (Passwort, TANs) erforderlich

Fristen für die Anmeldung und Modalitäten werden noch bekannt gegeben!

612200015	Prüfung INF015: Techniken des wissenschaftlichen Arbeitens	WS 07/08	13.02.2008	1,7 bestanden	4,0
612100021	Modul INF021: Lineare Algebra	SS 08		1,0 bestanden	8,0

Die üblichen Appelle

- ▶ Unterschätzen Sie nicht die Vorlesungsnachbereitung!
- ▶ Beschäftigen Sie sich aktiv mit dem Stoff!
- ▶ Setzen Sie Ihre Zeit gezielt ein!
- ▶ Bleiben Sie dran!
- ▶ Arbeiten Sie zusammen, aber schwimmen Sie nicht nur mit!
- ▶ Seien Sie in der Lage, Gruppenlösungen selbst zu präsentieren!
- ▶ Setzen Sie sich selbst an den Rechner zum Programmieren!
- ▶ Nehmen Sie Termine (und Bestimmungen) ernst!
- ▶ Fragen Sie!

Warum Programmieren/Programmiersprachen?

$$fib_0 = 1$$

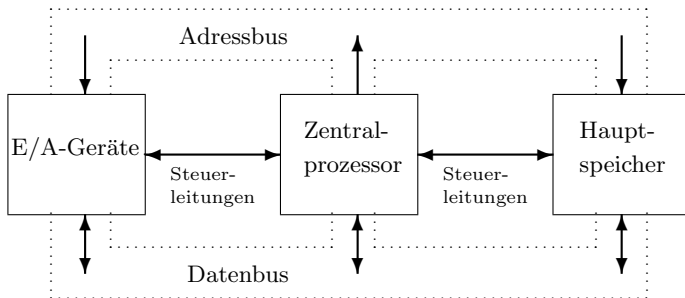
$$fib_1 = 1$$

$$fib_{n+2} = fib_n + fib_{n+1}$$

1, 1, 2, 3, 5, 8, 13, 21, ...



Von-Neumann-Rechner:



Warum Programmieren/Programmiersprachen?

$$fib_0 = 1$$

$$fib_1 = 1$$

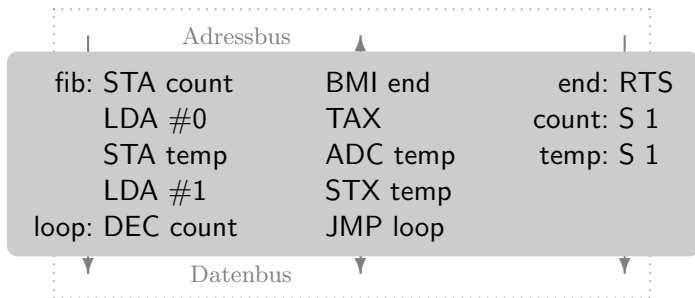
1, 1, 2, 3, 5, 8, 13, 21, ...

$$fib_{n+2} = fib_n + fib_{n+1}$$



? mittels einer Hochsprache!

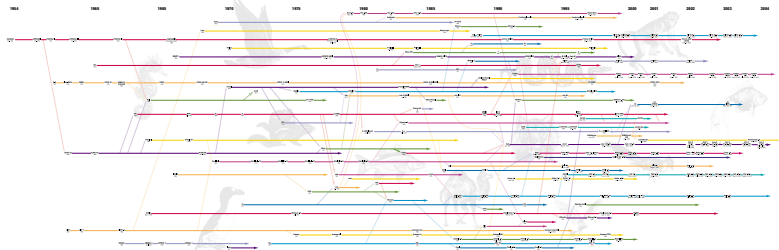
Von-Neumann-Rechner:



Eine Vielfalt an Höheren Programmiersprachen

History of Programming Languages

O'REILLY



www.oreilly.com

Copyright © 2004 by O'Reilly Verlag GmbH & Co. KG
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage and retrieval system, without the prior written permission of the publisher.

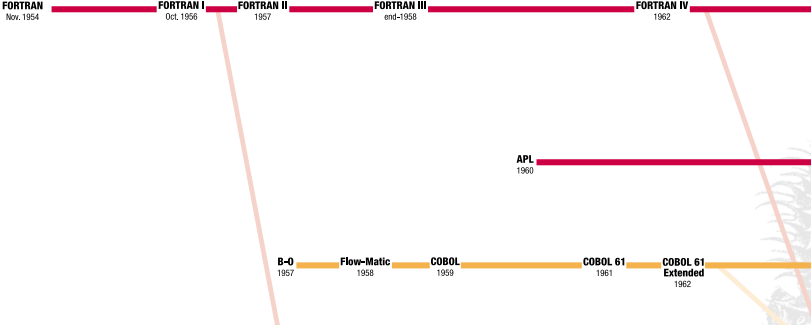
Copyright © 2004 by O'Reilly Verlag GmbH & Co. KG
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage and retrieval system, without the prior written permission of the publisher.



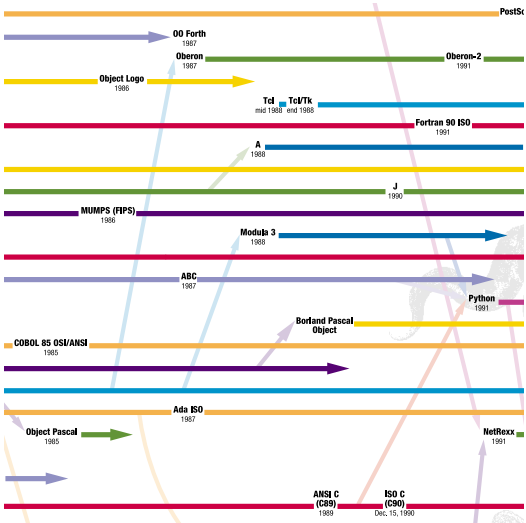
Eine Vielfalt an Höheren Programmiersprachen

1954

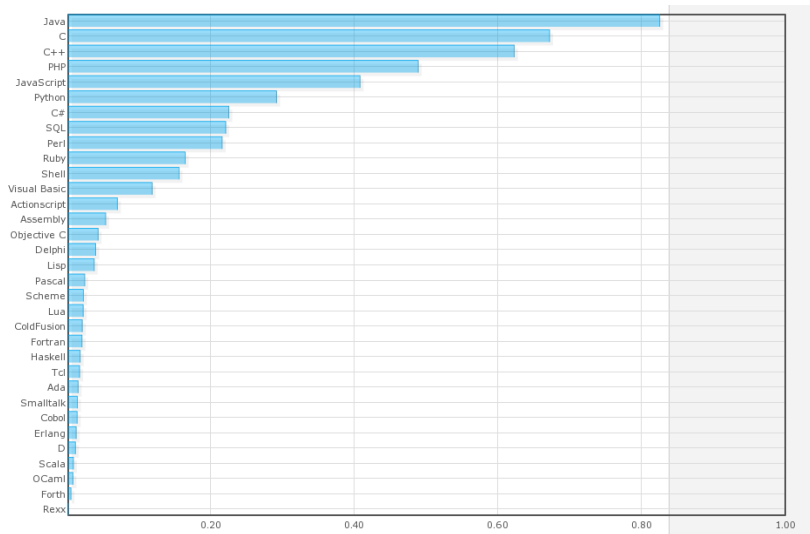
1960



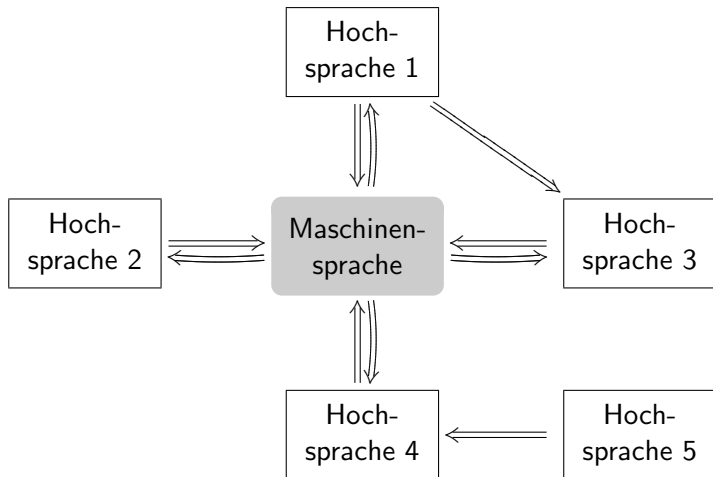
Eine Vielfalt an Höheren Programmiersprachen



<http://langpop.com/>



Übersetzung von Programmiersprachen



Warum Algorithmisches Denken?

Analogie zu mathematischer Modellbildung:

„Textaufgabe“

Wir beobachten das Wachstum einer Kaninchenpopulation. Zu Beginn gibt es ein neugeborenes Paar Kaninchen. Jedes Paar Kaninchen wirft pro Monat ein weiteres Paar Kaninchen. Allerdings bekommt ein neugeborenes Paar erst im zweiten Lebensmonat Nachwuchs. Kaninchen sind unsterblich. Wie entwickelt sich die Population?



„Rechenvorschrift“

$$fib_0 = 1$$

$$fib_1 = 1$$

$$fib_{n+2} = fib_n + fib_{n+1}$$

Warum Algorithmisches Denken?

Zur mechanischen Berechnung, weitere Betrachtungen notwendig:

„Rechenvorschrift“

$$fib_0 = 1$$

$$fib_1 = 1$$

$$fib_{n+2} = fib_n + fib_{n+1}$$



Beobachtung: Man braucht jeweils nur die Werte der vorangegangenen zwei Monate zu kennen.

Warum Algorithmisches Denken?

Zur mechanischen Berechnung, weitere Betrachtungen notwendig:

„Rechenvorschrift“

$$fib_0 = 1$$

$$fib_1 = 1$$

$$fib_{n+2} = fib_n + fib_{n+1}$$



Prinzipielles Vorgehen

1. Man merke sich jeweils den Wert des vorletzten und des letzten Monats.
2. Für jeden neuen Monat bilde man einen neuen Wert durch Addition der beiden bisher gemerkten.
3. Man „vergesse“ den Wert des vorletzten Monats und merke sich dafür den neuen Wert.

Warum Algorithmisches Denken?



Weiter konkretisierte Anweisungen

Um den Wert des n . Monats zu berechnen:

1. Verwende zwei „Variablen“ x und y , zu Beginn beide gleich 1.
2. Wenn $n < 2$, dann bereits fertig.
3. Andernfalls:
 - 3.1 Belege x und y neu mit y bzw. $x + y$.
 - 3.2 Das Ganze $n - 1$ mal.
4. Das Ergebnis liegt zum Schluss in y bereit.

Warum Algorithmisches Denken?



Weiter konkretisierte Anweisungen

Um den Wert des n . Monats zu berechnen:

1. Verwende zwei „Variablen“ x und y , zu Beginn beide gleich 1.
2. Wenn $n < 2$, dann bereits fertig.
3. Andernfalls:
 - 3.1 Belege x und y neu mit y bzw. $x + y$.
 - 3.2 Das Ganze $n - 1$ mal.
4. Das Ergebnis liegt zum Schluss in y bereit.



```
int fib(int n){  
    int x=1, y=1, i, z;  
    if (n>=2)  
        for (i=1; i<n; i=i+1) {z=x; x=y; y=z+y;}  
    return y;}  

```


Warum Algorithmisches Denken?

Es wären auch andere Realisierungen möglich gewesen, etwa:

„Rechenvorschrift“

$$fib_0 = 1$$

$$fib_1 = 1$$

$$fib_{n+2} = fib_n + fib_{n+1}$$



„Algorithmus“

Um den Wert des n . Monats zu berechnen:

1. Wenn $n < 2$, dann bereits fertig; das Ergebnis ist 1.
2. Andernfalls:
 - 2.1 Berechne separat den Wert des vorletzten Monats.
 - 2.2 Berechne separat den Wert des letzten Monats.
 - 2.3 Das Ergebnis ist die Summe dieser beiden Werte.

Warum Algorithmisches Denken?



„Algorithmus“

Um den Wert des n . Monats zu berechnen:

1. Wenn $n < 2$, dann bereits fertig; das Ergebnis ist 1.
2. Andernfalls:
 - 2.1 Berechne separat den Wert des vorletzten Monats.
 - 2.2 Berechne separat den Wert des letzten Monats.
 - 2.3 Das Ergebnis ist die Summe dieser beiden Werte.



```
int fib(int n){  
    if (n<2) return 1;  
    int x = fib(n-2);  
    int y = fib(n-1);  
    return x+y;} 
```

Einige Beobachtungen

- ▶ Die erste Variante zur Berechnung der Fibonacci-Zahlen ist deutlich effizienter.
- ▶ Der Unterschied liegt bereits in der algorithmischen Idee, nicht erst in der letztlichen Umsetzung als Programm begründet.

Algorithmenentwurf

- ▶ kreativer Akt
- ▶ „Kunst“
- ▶ schwer zu erlernen

vs.

Programmkonstruktion

- ▶ zum Teil „mechanisch“
- ▶ „Handwerk“
- ▶ leichter zu erlernen

- ▶ Jedes Problem lässt sich (wenn überhaupt) durch verschiedene Algorithmen lösen.
- ▶ Jeder Algorithmus lässt sich durch verschiedene Programme umsetzen.

Vom Problem zum Programm — Begrifflichkeiten

- Problem:**
- ▶ mit Rechnerhilfe zu lösende Aufgabenstellung
 - ▶ abstrakte Spezifikation des zu erreichenden Ziels
 - ▶ Spezifikation der Voraussetzungen

- Algorithmus:**
- ▶ möglichst präzise beschriebener Plan/Prozess
 - ▶ effektiv ausführbare Einzelschritte
 - ▶ möglichst effizient
 - ▶ oft informell repräsentiert
(Diagramm, natürliche Sprache, Pseudocode)

- Programm:**
- ▶ textuelle Repräsentation eines Algorithmus
 - ▶ präzisiert alle Details des Algorithmus
(so dass maschinell ausführbar)
 - ▶ zwingend in einer formalen Sprache abgefasst

Genauere Klärung des Begriffs „Algorithmus“

Algorithmen:

- ▶ ... sind durch endlichen Text vollständig beschrieben
(Finitheit)
- ▶ ... laufen in einzelnen, wohldefinierten Schritten ab
(Effektivität)
- ▶ ... legen eindeutig fest, welcher Schritt auf einen vorangegangenen folgt
(Determiniertheit)
- ▶ ... kommen nach endlich vielen Schritten zum Schluss
(Terminierung)

Jedoch weicht man einige dieser Bedingungen mitunter auf und spricht dann von:

- ▶ nicht-deterministischen Algorithmen
- ▶ nicht-terminierenden Algorithmen

Jenseits der formalen Definition von „Algorithmus“

Weitere, „wünschenswerte“, Eigenschaften von Algorithmen:

- ▶ Korrektheit (erfüllt vorgegebene Ein-/Ausgabespezifikation)
- ▶ Effizienz (verursacht möglichst wenig Aufwand)
- ▶ Verständlichkeit (kurzer Text, übersichtlich)
- ▶ Anpassbarkeit (leicht veränderbar)

Mitunter widersprechen sich diese erwünschten Eigenschaften gegenseitig, bzw. müssen gegeneinander abgewogen werden.

Wann gilt ein Algorithmus eigentlich als „korrekt“?

Partielle Korrektheit:

- ▶ Vorgabe einer genauen Ein-/Ausgabespezifikation:
 - ▶ Welche Eingabegrößen sind erforderlich?
 - ▶ Welchen Anforderungen müssen diese Eingaben genügen, damit der Algorithmus funktioniert?
 - ▶ Welche Ausgabegrößen — mit welchen Eigenschaften — werden erzeugt?
- ▶ Jede erzeugte Ausgabe genügt der Ausgabespezifikation, sofern die Eingaben der Eingabespezifikation genügt haben.

Totale Korrektheit: Partielle Korrektheit + Terminierung

Stets totale Korrektheit zu garantieren wäre ideal, jedoch:

- ▶ Beweise können sehr anspruchsvoll werden, und sind im Allgemeinen nicht mechanisierbar.
- ▶ In der Praxis erfolgt (leider) oft nicht einmal eine vollständige Ein-/Ausgabespezifikation.

Der verantwortungsbewusste Ingenieur . . . testet.

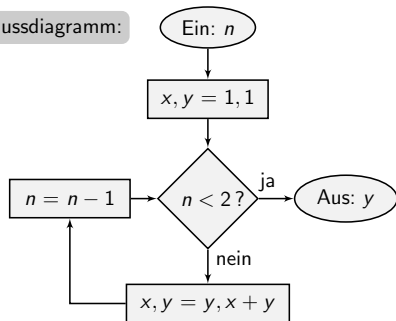
Verschiedene Beschreibungsmöglichkeiten

Natürlichsprachlicher Ablaufplan:

Um den Wert des n . Monats zu berechnen:

1. Belege zwei Variablen x und y jeweils mit 1.
2. Wenn $n < 2$, dann bereits fertig.
3. Andernfalls:
 - 3.1 Belege x und y neu mit y bzw. $x + y$.
 - 3.2 Das Ganze $n - 1$ mal.
4. Das Ergebnis liegt zum Schluss in y bereit.

Flussdiagramm:



Programm:

```
int fib(int n){
    int x=1, y=1, i, z;
    if (n>=2)
        for (i=1; i<n; i=i+1) {z=x; x=y; y=z+y;}
    return y;}
```


Eine konkrete Programmiersprache: C



- ▶ 1972 entwickelt von Dennis Ritchie an den Bell Labs
- ▶ entscheidend für die Entwicklung von UNIX
- ▶ maschinennah, explizite Speicherverwaltung
- ▶ trotzdem portabel, weit verbreitet
- ▶ einfacher Aufbau, geringe Anzahl verschiedener Konzepte
- ▶ imperativ!

Genaugenommen: „Disciplined C“

Vereinfachungen/Einschränkungen für die Vorlesung:

- ▶ Verzicht auf einige Kontrollstrukturen
- ▶ keine Pointer-Arithmetik
- ▶ etwas strengere Typdisziplin
- ▶ keine Makros/Präprozessor

Fokus der Vorlesung auf:

- ▶ imperative Kontrollstrukturen
- ▶ allgemeine Programmierkonzepte
- ▶ algorithmische Prinzipien
- ▶ wichtige Datenstrukturen

Nicht, zum Beispiel:

- ▶ Modularisierung
- ▶ Perlen der Algorithmik

1.		Logik u. Diskrete Strukturen	Technische Informatik	Informationssysteme	Imperative Programmierung	Techniken des wissenschaftlichen Arbeitens	28/ 30
2.	Analysis	Lineare Algebra	Systemnahe Informatik		Objektorientierte Softwareentwicklung		30
3.	Angewandte Mathematik	Algorithmen u. Berechnungskomplexität I		Software-technologie	Systemnahe Programmierung		30/ 28
4.		Algorithmen u. Berechnungskomplexität II			Wahlpflicht: Deskriptive Programmierung		6
5.						Projektgruppe - Seminar - Praktikum	
6.						Bachelormodul - B.arbeit - B.seminar	

+ mehrere Abhängigkeiten bei Nebenfach- und Wahlpflichtmodulen

Syntaxbeschreibung

Zur Erinnerung, (Teil von einem) C-Beispielprogramm:

```
int fib(int n){  
    int x=1, y=1, i, z;  
    if (n>=2)  
        for (i=1; i<n; i=i+1) {z=x; x=y; y=z+y;}  
    return y;}  

```

Wichtige Fragen:

- ▶ Aus welchen „Bausteinen“ wird ein Programm zusammengesetzt?
- ▶ Nach welchen Regeln werden diese systematisch kombiniert?
- ▶ Wie lässt sich dies formal beschreiben (und intuitiv verstehen)?
- ▶ Wie kann ein Compiler an Hand einer solchen Beschreibung feststellen, ob ein Programm überhaupt legal ist?

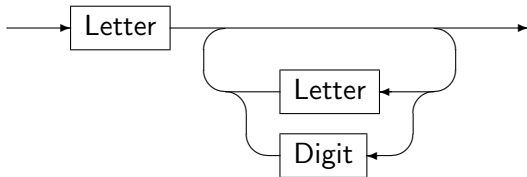
Syntaxbeschreibung

Formale Sprachtheorie kennt verschiedene Beschreibungsmethoden:

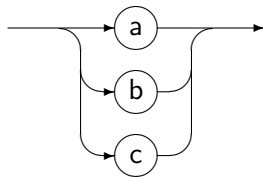
- ▶ formale Grammatiken
- ▶ endliche Automaten
- ▶ **Syntaxdiagramme**

Ein einfaches Beispiel:

Ident



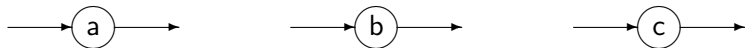
Letter



Syntaxdiagramme

Allgemeiner Aufbau:

- ▶ Ovale:



- ▶ Rechtecke:



- ▶ Verbindungen (mit Richtungspfeilen):

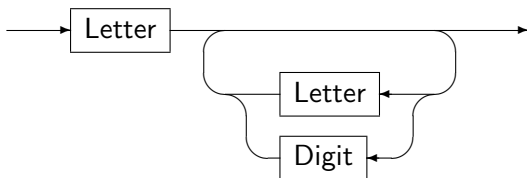
- ▶ direkte Linien
- ▶ Verzweigungen
- ▶ Zusammenführungen

Syntaxdiagramme

In einem **System** von Syntaxdiagrammen:

- ▶ Jedes Syntaxdiagramm hat einen eindeutigen Namen, genau einen Anfang und genau ein Ende.
- ▶ Jedes Rechteck ist mit dem Namen eines Syntaxdiagramms beschriftet.
- ▶ Jedes Oval ist mit einem Symbol (der Zielsprache) beschriftet.
- ▶ Jedes Rechteck/Oval hat genau einen Eingang und genau einen Ausgang.

Ident



Letter

