Einführung in die Programmierung

MIT DER PROGRAMMIERSPRACHE C

PROF. DR. THOMAS GABEL

Willkommen zur Vorlesung!

Termine:

- Vorlesung donnerstags, 11.45 – 13.15 Uhr, in Raum 4-109/110
- vier Übungsgruppen, ab KW43 (d.h. ab nächster Woche!)
  - montags, 14.15 – 15.45 Uhr, in Raum BCN-305, Start: 20.10.2025
  - mittwochs, 8.15 – 9.45 Uhr, in Raum 1-236, Start: 22.10.2025
  - mittwochs, 11.45 – 14.15 Uhr, in Raum 1-236, Start: 22.10.2025
  - donnerstags, 14.15 – 15.45 Uhr, in Raum 1-248, Start: 23.10.2025
  - Anwesenheitspflicht (mindestens 80%)

Personen:

- Dozent: Prof. Dr. Thomas Gabel (VL+3xÜ)
- Wiss. Mitarbeiter: Dominic Gibietz (1xÜ)
- Tutoren: Roman Ahmad, Emre Özöner, Gina Romanazzi

Kontakt:

- eMail: tgabel@fra-uas.de und dominic.gibietz@fra-uas.de
- Raum 1-202, Telefon: 069/1533-2538 sowie 0179/4392368
- http://www.frankfurt-university.de/tgabel sowie http://www.tgabel.de

EifP: Lernziele

Überblick über Software-Entwicklung und Programmieren

- Wie komme ich vom Algorithmus zum Programm?
- Welche Sprache versteht mein Rechner?

Technische und formale Grundlagen der Programmierung

- Mit welchen Gestaltungsmitteln kann ich den Programmentwurf unterstützen?
- Wie bringe ich gut nachvollziehbare und erweiterbare Struktur in ein Programm?

Darauf aufbauend in den folgenden Vorlesungseinheiten:

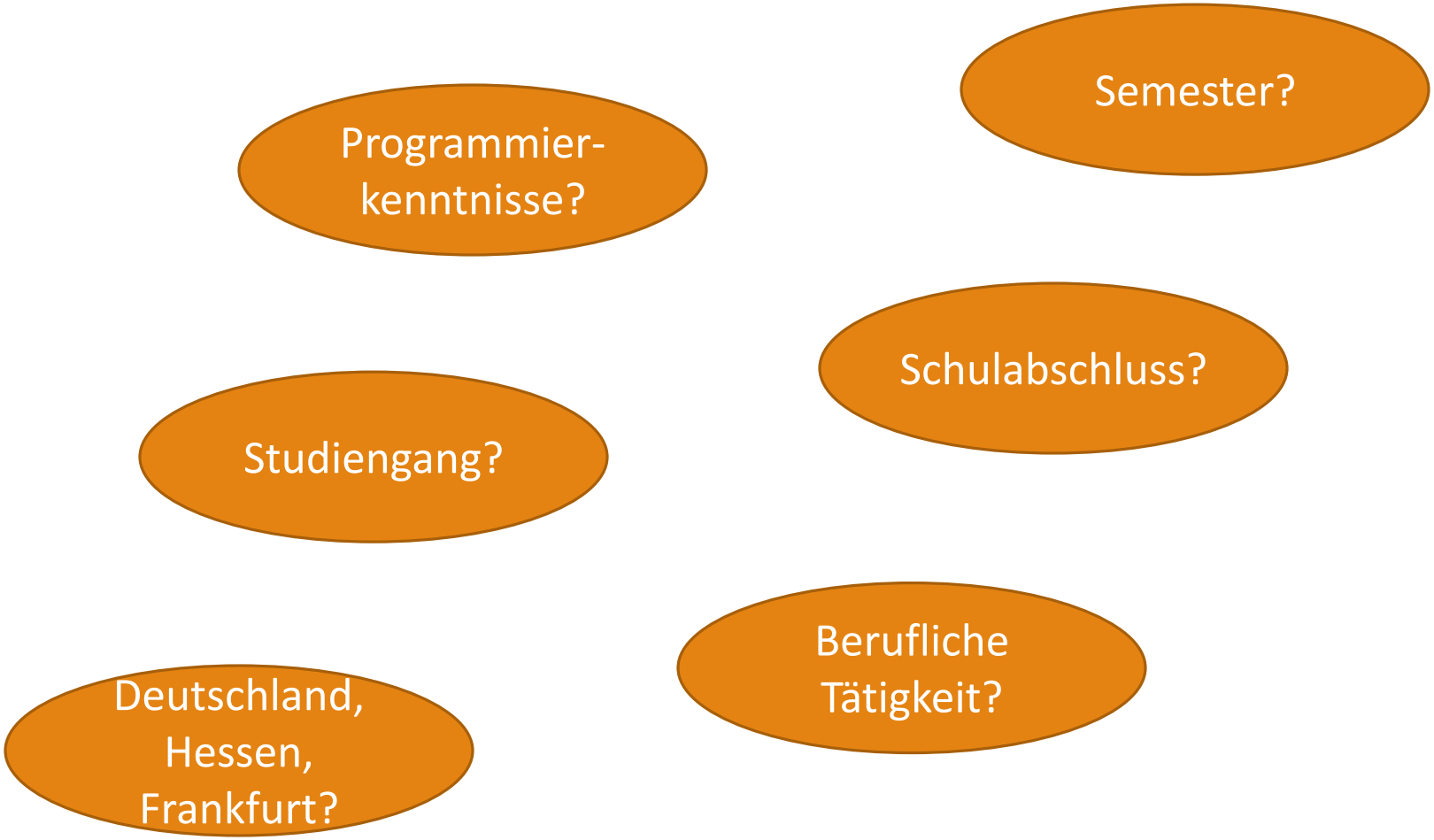
- Welche (programmier-)sprachlichen Mittel gibt es, um Programme strukturiert zu entwickeln?
- Wie werden (Kontroll-)Strukturen in der Programmiersprache C umgesetzt?

Es gilt:

- Komponisten müssen Klavier spielen können!
- Architekten müssen zeichnen können!

Daher:
Informatiker müssen programmieren können!

Und Sie?


Semester?
Programmier-
kenntnisse?
Schulabschluss?
Studiengang?
Berufliche
Tätigkeit?
Deutschland,
Hessen,
Frankfurt?

Organisatorisches

Vorlesungsziele und -besonderheiten

- Sie lernen Grundkonzepte der imperativen Programmierung.
- Sie proben diese Konzepte aus und erlernen die Programmiersprache C.
- Wir arbeiten durchgängig unter GNU/Linux.
  - Die „GNU Compiler Collection“ (GCC) wird verwendet.
  - Ggf. für Zuhause: GCC ist auch für Windows / MacOS verfügbar.
- Alle Beispiele, Anleitungen und Erklärungen sind auf diese Rahmenbedingungen hin angepasst.

campUAS-Kurs

- Name:
  „Gabel: Einführung in die Programmierung - WS 25/26“
- Einschreibeschlüssel: **`9iJnHz&`**
  - **Auch wenn Sie bereits eingetragen waren, müssen Sie sich mit jenem Schlüssel noch einmal neu einschreiben.**
- Folien, Übungsblätter etc.
  zu gegebener Zeit verfügbar

Bestandteile der Lehrveranstaltung

Vorlesung
- regelmäßige Teilnahme
- aktives Mitschreiben
- Fragen stellen

Übungen
- regelmäßige Teilnahme
- Aufgaben im Vorfeld bearbeiten
- Teamarbeit & Lösungspräsentation

Ihr Engagement!
- Vor- und Nachbereitung der Vorlesung
- Übungsaufgaben zu Hause lösen
- Umsetzen eigener Programmierideen

erfolgreiche Klausur im Februar
(eigenständige Programmierung!)

Organisatorisches

Übungen

- jede Woche gibt es ein Aufgabenblatt zur individuellen Bearbeitung
- Zielstellung:
  - Bearbeitung im Zweierteam oder alleine
  - Bearbeitung im Vorfeld der Übung (zumindest teilweise)
  - fehlende Teile werden in der Übungsstunde (ggf. mit Unterstützung) gelöst
- Inhalte der Übungsstunden
  - Besprechung von (weiteren) Inhalten aus der Vorlesung
  - Unterstützung bei der Anfertigung der Lösungen zu den Übungsaufgaben durch Professor und Tutoren
  - Erläuterungen von (Teil-)Lösungen der zu bearbeitenden Aufgaben
  - Vorstellung gelöster Aufgaben durch Übungsteilnehmer (an Tafel / Beamer)
  - Vorführung gelöster Aufgaben im 4(6)-Augen-Gespräch
  - Beantwortung allgemeiner Fragen (auch zu Vorlesungsinhalten)
- Achtung: **Anwesenheitspflicht (>80%)** in den Übungsstunden
  - Vermerk auf Anwesenheitsliste nicht vergessen!
- Übungen starten ab 20.10.2025 (für Erstsemester! höhere Semester ab Folgewoche!)
  - Übungsblatt 1: Login in den Rechnerräumen, Linux-Einführung, Übungsblätter auf campUAS-Seite des Kurses

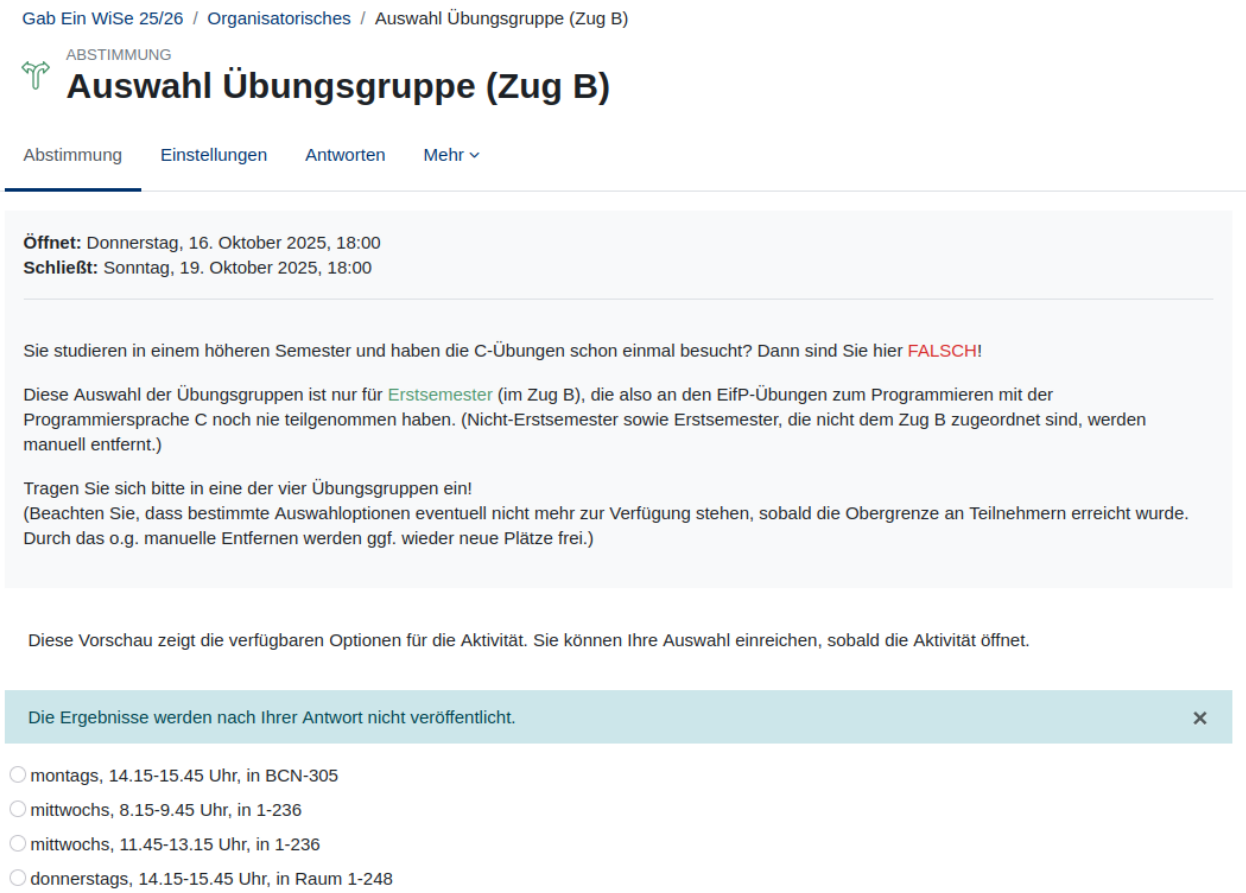
Organisatorisches

Übungsgruppen

- Eintragung in Übungsgruppen im campUAS-Kurs zur Vorlesung
- Beginn der Eintragung: Donnerstag, der 16.10.2025, ab 18 Uhr
- Ende der Eintragung: Sonntag, der 19.10.2025, 18 Uhr

Gruppenzuordnung ist verbindlich für Übungen!
Ü-Start: 20.10.2025!

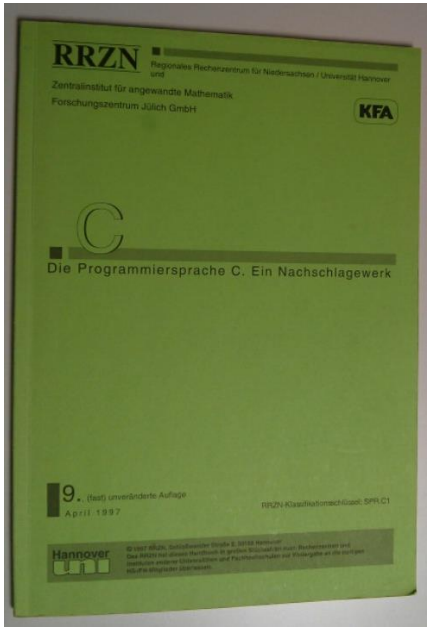
Anwesenheitserfassung ab 20.10.2025!

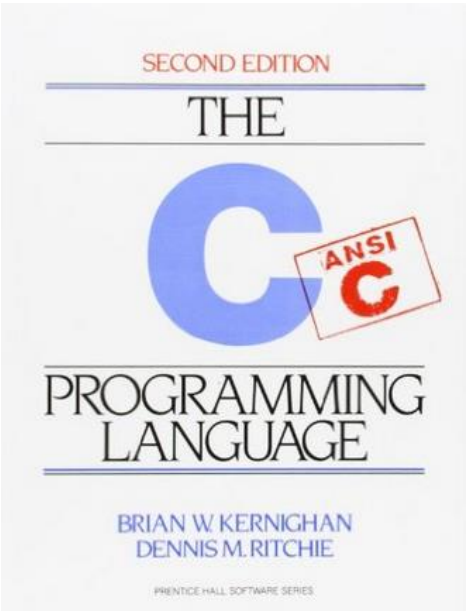


Literatur / Bücher

Literatur

- Vorlesungsfolien
- Es gibt kein „offizielles“ Buch zur Vorlesung, alle Details werden in der Vorlesung besprochen, bzw. können sich im Selbststudium angeeignet werden.
- lesenswerte Bücher
  - Regionales RZ Niedersachsen (RRZN):
    „Die Programmiersprache C. Ein Nachschlagewerk“, 3.80 Euro
    - nur als Gebrauchtexemplar verfügbar
  - Kerningham/Ritchie: „Programmieren in C“
  - Unix-Man-Pages für C-Funktionen
- weitere Literatur
  - Dmitrovic: „Modern C for Absolute Beginners“
    - schlank gehalten
    - gut als Referenz für Einsteiger geeignet
  - Gustedt: „Modern C“
    - online verfügbar als PDF, leicht fortgeschritten
    - https://gustedt.gitlabpages.inria.fr/modern-c/





Überblick über die Vorlesung

1. Algorithmen, Programme und Software
2. Einstieg in die Programmierung in C
3. Strukturiertes Programmieren in C
4. Effizientes Programmieren in C
5. Fortgeschrittene Aspekte der Programmierung in C

Einführung in die Programmierung

MIT DER PROGRAMMIERSPRACHE C

PROF. DR. THOMAS GABEL

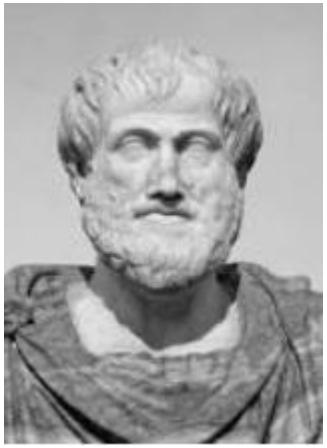
Überblick über die Vorlesung

1. **Algorithmen, Programme und Software**
2. Einstieg in die Programmierung mit C
3. Strukturiertes Programmieren in C
4. Effizientes Programmieren in C
5. Fortgeschrittene Aspekte der Programmierung in C

1. Algorithmen, Programme und Software

Softwaresysteme (1)

Definition: Ein **System** besteht aus Teilen (Komponenten, Subsystemen), die in geordneter Weise miteinander in Beziehung stehen.



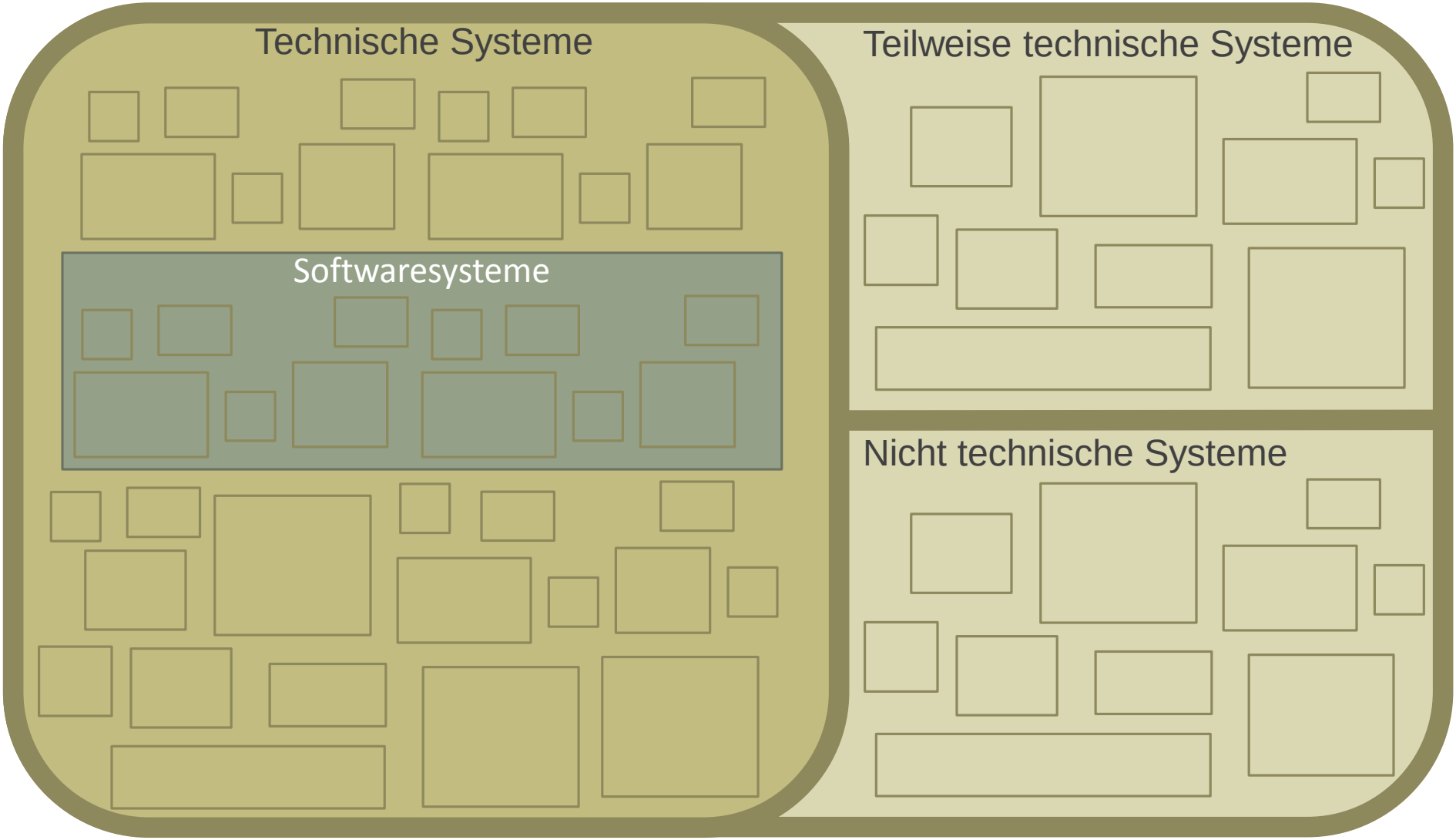
Zitat:
„Das Ganze ist mehr als die Summe seiner Teile.“
Aristoteles

Praktisch unterscheidet man zwischen

- technischen Systemen
  - Kraftwerk, Rechner, Auto, Telefonnetz
- teilweise technischen Systemen
  - Verkehrssystem, Bibliotheksverwaltungssystem
- nicht technischen Systemen
  - Rechtssystem, Wirtschaftssystem, Ökosystem, Sonnensystem

Definition: Ein **Softwaresystem** ist ein technisches System oder ein Teilsystem eines technischen Systems, bei dem die Funktionalität mittels Software realisiert ist.

Überblick Systeme



Softwaresysteme (2)

Frage: Welche Softwaresysteme kennen Sie?

- Betriebssystem
- Fenstersystem
- Dateisystem
- Computerspiel
- Textverarbeitungssystem
- Internet / World Wide Web
- Informations- und Buchungssystem der Bahn
- Telekommunikationssystem
- eingebettete Softwaresysteme (z.B. im Auto)
- …

Frage: Welche Eigenschaften sind all diesen Systemen gemein?

Antwort: nächste Folien

Softwaresysteme (3)

Beobachtung: Softwaresysteme sind nicht physisch.

- lassen sich nicht anfassen und nicht direkt betrachten
- sind konstruierte, technische Systeme, d.h. nicht natürlichen Ursprung

Beobachtung: Softwaresysteme dienen unterschiedlichen Zwecken.

- z.B. als Plattform für andere SW-Systeme (Betriebssystem)
- z.B. der Kommunikation mit Menschen (Mobilfunksystem)
- z.B. der Steuerung von Maschinen (Autopilot)

Beobachtung: Softwaresysteme sind bausteinartig.

- können auf anderen Systemen aufbauen (Computerspiel → Betriebssystem)
- haben Schnittstellen zur Umgebung (HCI, Human Computer Interface)
- sind Teil komplexer Systeme (Auto)

Softwaresysteme (4)

Beobachtung: Softwaresysteme sind ausgesprochen heterogen.

- Größe (in Programmzeilen, in Entwicklerjahren)
- Terminierung (terminierend vs. nicht terminierend)
- Persistenz (Daten werden zwischen zwei aufeinanderfolgenden Benutzungen gespeichert)
- Interaktionsverhalten (z.B. interaktiv, reaktiv)
- Verteilung (lokal auf einem Rechner vs. über mehrere Rechner verteilt)
- Komplexität
- Qualitätseigenschaften (Stabilität, Korrektheit, Benutzerfreundlichkeit)

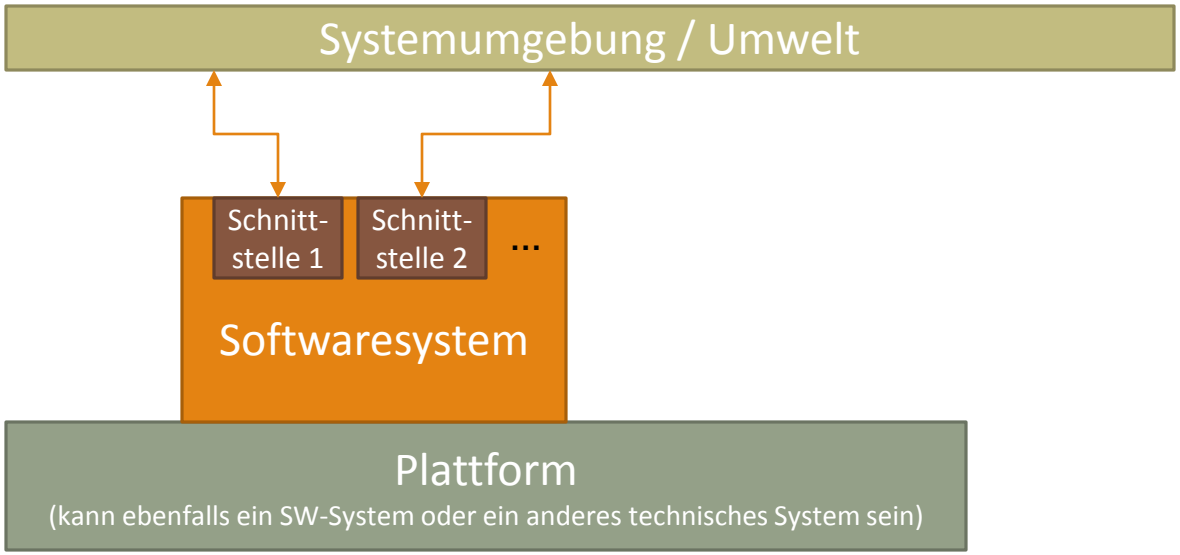
Beobachtung: Softwaresysteme spielen eine wichtige Rolle für die wirtschaftliche und gesellschaftliche Entwicklung.

- Effizienz, Automatisierung
- Herstellung konkurrenzfähiger Produkte
- Kommunikationsinfrastruktur
- Zugriff auf Wissen und Informationen

Softwaresysteme (5)

Bemerkungen:

- Jedes Softwaresystem läuft auf einer Plattform.
  - Dies kann eine Hardware- oder Softwareplattform sein.
- Softwaresysteme besitzen Schnittstellen zur Umgebung, d.h. einige (Software-)Teile dienen der Kommunikation mit anderen Systemen.
  - Bedienschnittstellen
  - Ein-/Ausgabeschnittstellen
  - Programmierschnittstellen



Software

Erinnerung: Definition: Ein **Softwaresystem** ist ein technisches System oder ein Teilsystem eines technischen Systems, bei dem die Funktionalität mittels Software realisiert ist.

Definition: Unter **Software** (Standarddefinition) versteht man

- Programme zur Beschreibung des erwünschten Systemverhaltens
- Daten zur Beschreibung von Informationen

Bemerkung:

- Um von einem Rechner (Computer) verarbeitet werden zu können, müssen die Beschreibungen in einer formalen Syntax (Programmiersprache) abgefasst sein.

Definition: Unter **Software** (erweiterte Definition) versteht man neben den Programmen und Daten auch Beschreibungen, die für die Erzeugung und Ausführung eines Softwaresystems nicht direkt notwendig sind. Dazu gehören

- Dokumentationen, Architekturbeschreibungen
- Daten für Anwendungs- und Testfälle
- Installations- und Wartungssoftware

Programme

Wenn es um Programmierung geht, stehen Programme im Mittelpunkt des Interesses; Programme sind ein wesentlicher Bestandteil von Software.

Definition: Ein **Programm** (Software-Definition) ist ein Text, der

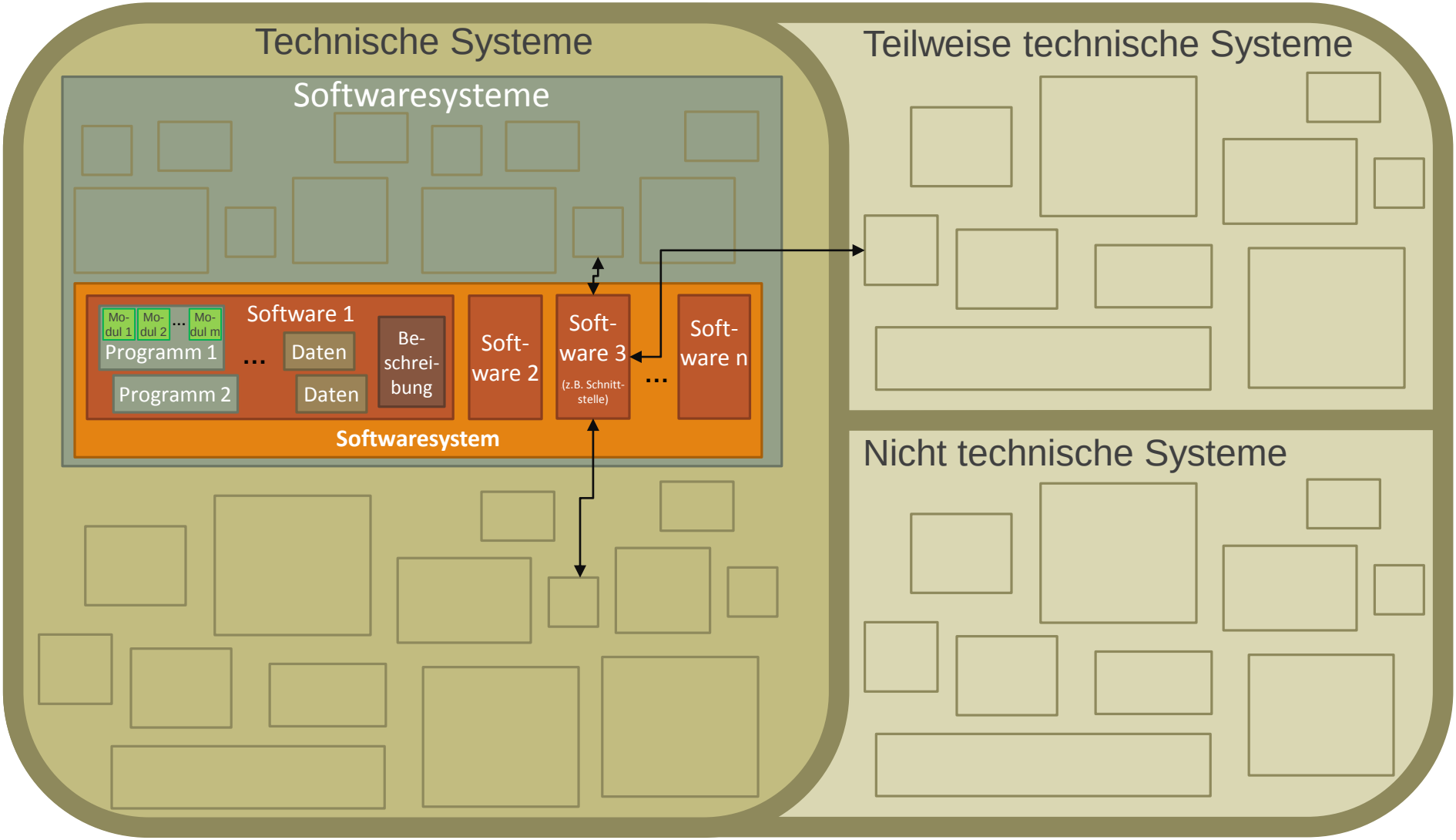
- in einer Programmiersprache verfasst ist und
- der hinreichend vollständig ist, um auf einem Rechensystem ausgeführt werden zu können.
- Einzelteile, aus denen ein Programm zusammengesetzt ist, werden auch Module genannt.

Beispiele:

- Programme → Texte, die in C, C++, Java, Pascal, Basic, Smalltalk, Python … verfasst sind
- Daten → Zahlen, Webseiten (HTML-Seiten), Latex-Dokumente, Excel-Tabellen
- Vermischung von Daten und Programmen → Webseiten mit eingebettetem Programmcode (z.B. Javascript in einer HTML-Seite)

Gesamtbild Softwaresystem:

Überblick Systeme



1. Algorithmen, Programme und Software

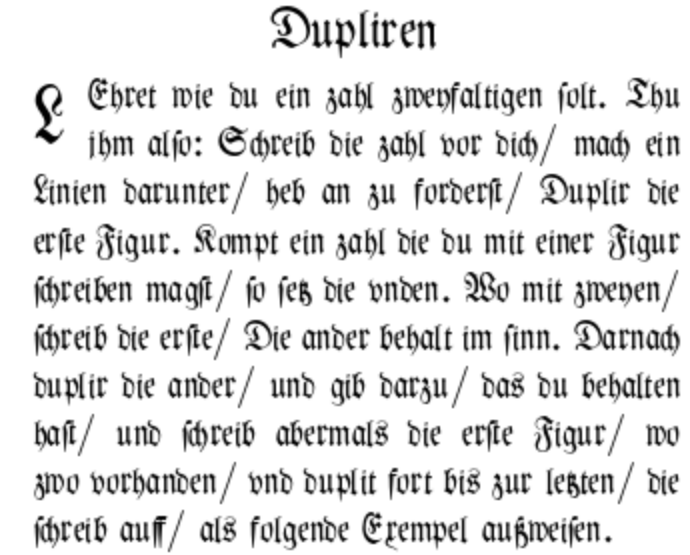
1. Vom System zum Programm
2. **Vom Algorithmus zum Programm**
3. Programmierparadigmen
4. Algorithmische Grundkonzepte
5. Strukturiertes Programmieren
6. Zusammenfassung & Ausblick

Ein Algorithmus …

```
Thu ihm also:
Schreib die zahl vor dich
mach ein Linien darunter
heb an zu forderst
Duplir die erste Figur.
Kompt ein zahl die du mit einer Figur /
  schreiben magst
so setz die unden.
Wo mit zweyen
schreib die erste
Die ander behalt im sinn.
Darnach duplir die ander
und gib darzu
das du behalten hast
und schreib abermals die erste Figur
wo zwo vorhanden
und duplir fort bis zur letzten
die schreibe ganz aus
als folgende Exempel ausweisen.
```

Was bewirkt dieser Algorithmus?

„Dupliren“
nach Adam Riese (1574):
„Lehret wie du ein zahl zweyfaltigen solt.“

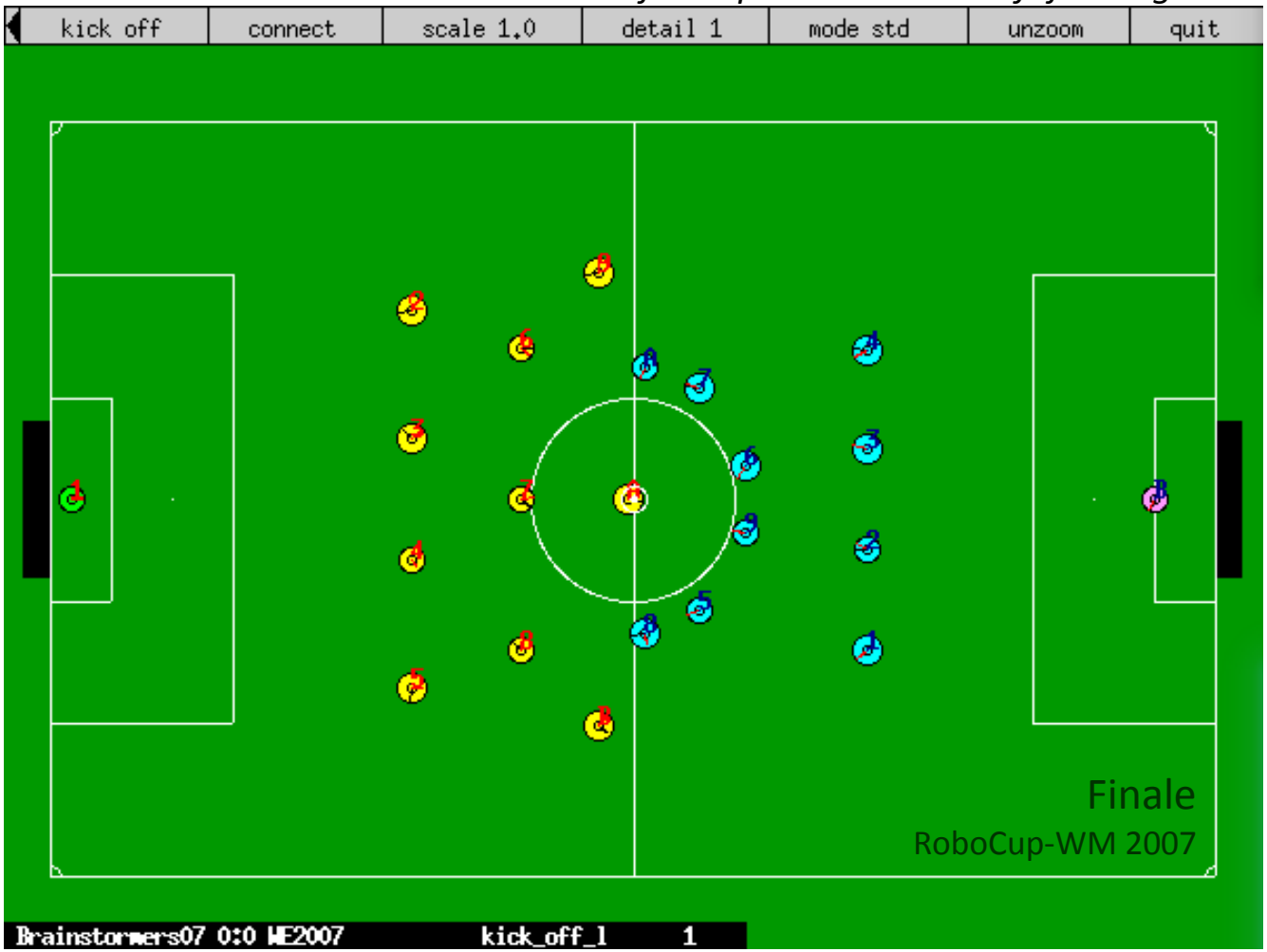


Dupliren

Lehret wie du ein zahl zweyfaltigen ſolt. Thu ihm alſo: Schreib die zahl vor dich/ mach ein Linien darunter/ heb an zu forderſt/ Duplir die erſte Figur. Kompt ein zahl die du mit einer Figur ſchreiben magſt/ ſo ſetz die vnden. Wo mit zweyen/ ſchreib die erſte/ Die ander behalt im ſinn. Darnach duplir die ander/ und gib darzu/ das du behalten haſt/ und ſchreib abermals die erſte Figur/ wo zwo vorhanden/ vnd duplit fort bis zur letzten/ die ſchreib auff/ als folgende Exempel außweiſen.

Weitere Algorithmen …

Fußballspielende Roboterfußballagenten



Vom Algorithmus zum Programm (1)

Definition: Ein **Algorithmus** ist ein Verfahren zur schrittweisen **Ausführung** von **(Berechnungs-)Abläufen**, das sich präzise und endlich beschreiben lässt.

- Beispiele: Kochrezept, Gleichungssystem lösen

Al-Chwarizmi, der Namensgeber des Algorithmus, auf einer sowjetischen Briefmarke anlässlich seines 1200-jährigen Geburtsjubiläums 1980

Hierbei soll

- die Beschreibung aus wohlverstandenen, ausführbaren Einzelschritten bestehen
- in einem Schritt eine oder mehrere Aktionen (ggf. parallel) ausgeführt werden
- jede Aktion von einem Zustand in einen Nachfolgezustand überführen.

Erinnerung:

- *Der Begriff **Software** bezeichnet eine Sammlung von Programmen, zugehörigen Daten sowie Dokumentation.*

Definition: Ein **Programm** (algorithmische Definition) bezeichnet Algorithmen, die so formuliert sind, dass sie auf einem Rechner ausgeführt werden können.

- Aber: Welche Sprache versteht der Rechner?

Vom Algorithmus zum Programm (2)

Bemerkungen zur Herkunft des Begriffes „Algorithmus“:

In dem Wort „Algorithmus“ lebt der Name des Universalgelehrten Abu Jafar Muhammad Ibn Musa Al-Chwarizmi (783-850 u.Z.) aus der in Mittelasien gelegenen Landschaft Choresmien fort, welcher etwa seit dem Jahr 800 an der Akademie der Wissenschaften („Haus der Weisheit“) in Bagdad - zusammen mit anderen Gelehrten - indische und griechische wissenschaftliche Schriften ins Arabische übersetzte und auf dieser Grundlage selbst weiter forschte.

Er schrieb mathematische und astronomische Lehrbücher, unter anderem ein weit verbreitetes und einflussreiches Mathematikwerk mit dem Titel Kitab Al-Jabr Wal-Muqabala, d.h. „Buch über die Rechnung durch Vergleich und Reduktion“, welches im 13. Jahrhundert ins Lateinische übersetzt wurde. In der Übersetzung beginnen die Kapitel jeweils mit Dixit Algorithmi, d.h. „Also sprach Al-Khwarizmi!“.

Quelle: R. Baumann: „Informatik“, Band 1

Programmiersprachen

Maschinen-sprachen | Assembler-Sprachen | Höhere Program-miersprachen

„low-level“

maschinenabhängig

bestehen aus Zahlenfolgen

```
11010011000111010100
10110101010111011000
01111010110010101101
01010101111011111101
01101010110001100101
```

vom Rechner direkt verarbeitbar

- d.h. ohne vorherigen Übersetzungsvorgang

schlecht lesbar

- aber: maximale Kontrolle des Entwicklers über Speicher- und Ausführungseffizienz

Programmiersprachen

Maschinen-sprachen | **Assembler-Sprachen** | **Höhere Program-miersprachen**

„middle“-level

maschinenabhängig

Maschinensprache wird in einer für Menschen lesbaren Form repräsentiert

- dennoch sehr hohe Kontrolle über Speicher- und Ausführungseffizienz

nur nach Übersetzung in Maschinensprache verarbeitbar

```
mov a1, 61h
mov ah, 09h
int 21h
mov ax, DATA
```


```
11010011000111010100
10110101010111011000
01111010110010101101
01010101111011111101
```

Programmiersprachen

Maschinen-sprachen	Assembler-Sprachen	Höhere Program-miersprachen

high-level

problemorientierte Sprachkonstrukte

rechnerunabhängig

vor Ausführung Übersetzung erforderlich

gut lesbar

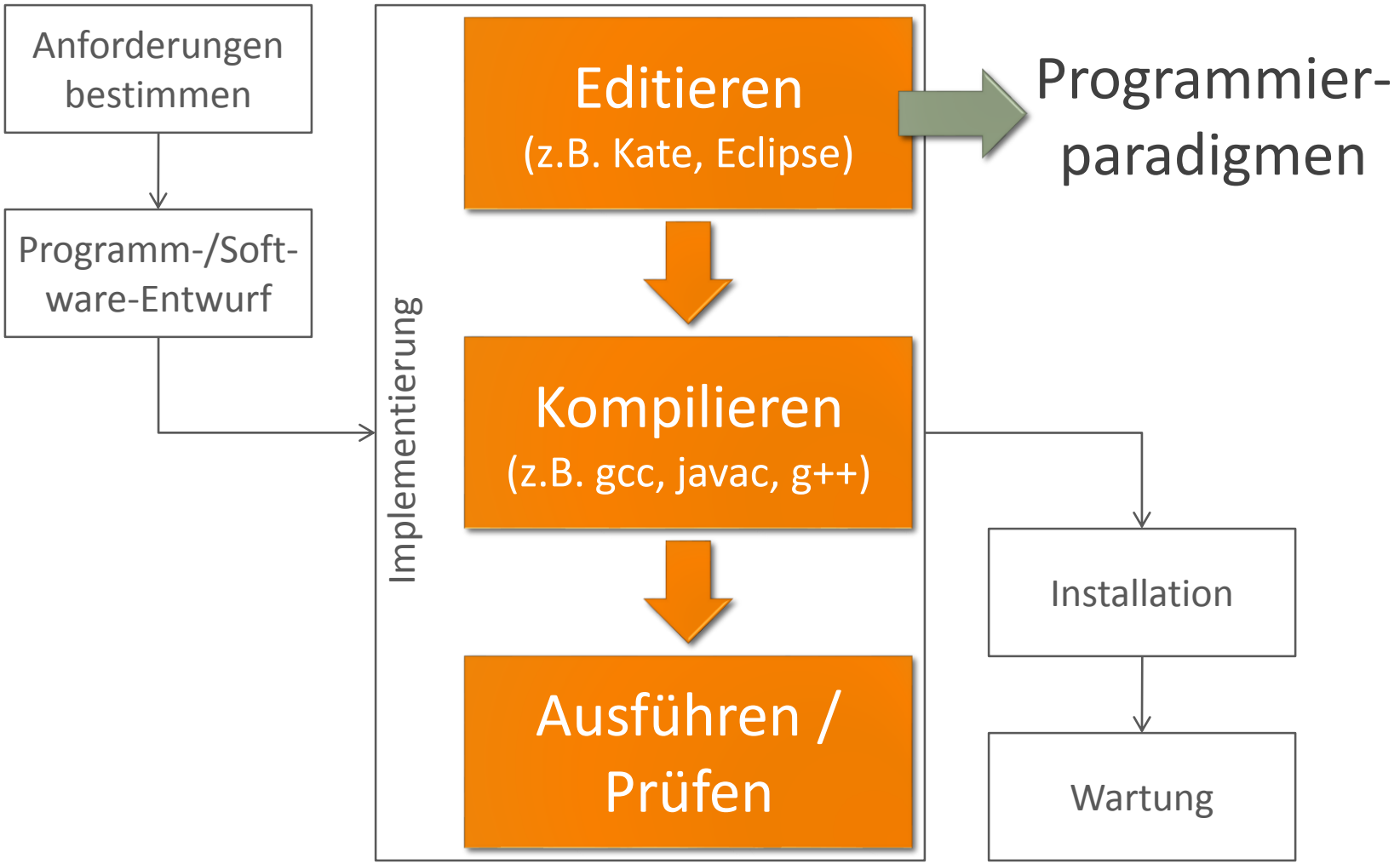
Effizienz von untergeordneter Bedeutung

```
z = x + y;
x = x - 2;
```

Compiler

```
11010011000111010100
10110101010111011000
01111010110010101101
01010101111011111101
```

Software-/Programmerstellung



1. Algorithmen, Programme und Software

1. Vom System zum Programm
2. Vom Algorithmus zum Programm
3. **Programmierparadigmen**
4. Algorithmische Grundkonzepte
5. Strukturiertes Programmieren
6. Zusammenfassung & Ausblick

Programmierparadigmen (1)

Definition: Die **Paradigmen der Programmierung** sind charakterisiert durch das Zusammenwirken bestimmter Konzepte, Vorgehensweisen, Techniken, Theorien und Standards.

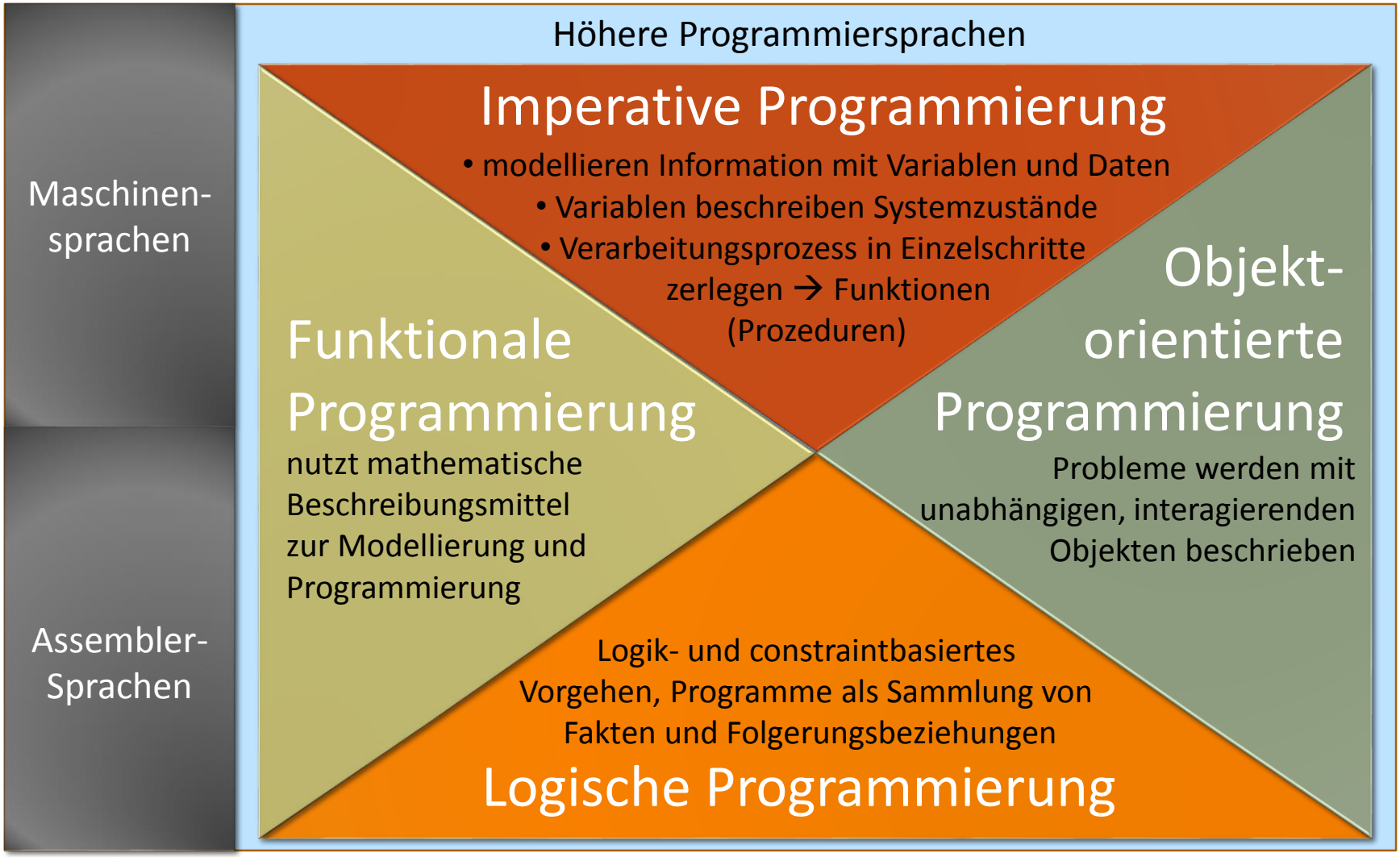
- Sich für ein Programmierparadigma zu entscheiden bedeutet also, sich festzulegen, welche konkreten Techniken, Konzepte und theoretischen Grundlagen man bei der Programmerstellung verwenden will.
- Bestimmte Programmiersprachen unterstützen bestimmte (eines oder mehrere) Programmierparadigmen.
  - z.B. C → 1, Java → 2-3

In der Softwareentwicklung und Programmierung unterscheiden wir vier Programmierparadigmen

- imperative (prozedurale) Programmierung
- objektorientierte Programmierung
- funktionale Programmierung
- logische Programmierung

(Anmerkung: Zu den einzelnen Paradigmen gibt es diverse Erweiterungen, so dass diese Zerlegung zum Teil kontrovers diskutiert wird.)

Programmierparadigmen (2)



Beispiel: Logische Programmierung

Kernideen:

- verwende mathematische und logische Beschreibungsmittel zur Modellierung und zur Programmierung
- Programm ist eine Ansammlung von Fakten und Folgerungsbeziehungen

Beispiel in Prolog:

```
istProfessor( gabel ).
istMensch( aristoteles ).
istMensch( X ) :-- istProfessor( X );
istSterblich( X ) :-- istMensch( X );
```

- Anwendung eines logischen Programms wird durch Anfragen (Queries) ausgelöst:

```
istMensch( aristoteles )?
istSterblich( gabel )?
istSterblich( X )?
istSterblich( mozart )?
```

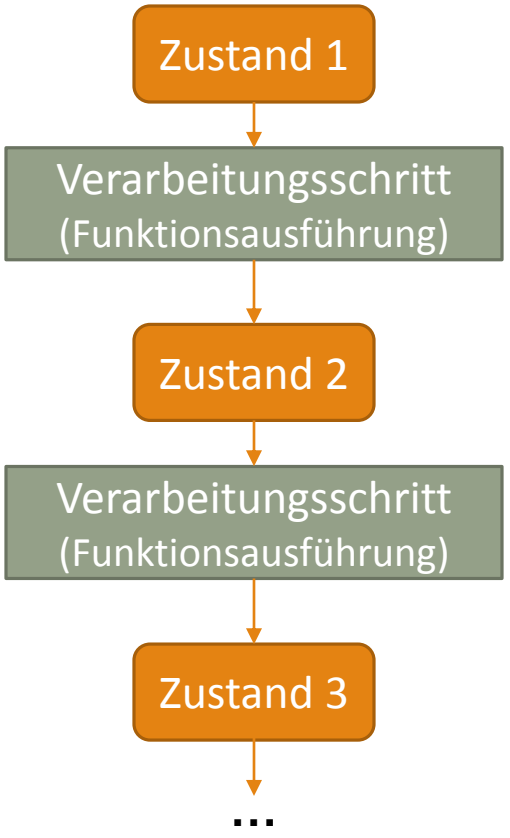
- Stärke: mathematischer Begriffsbildung kann genutzt und von Verarbeitungsschritten abstrahiert werden, Modellierung wird aber erschwert

Imperative Programmierung

Schwerpunkt dieser Vorlesung (imperatives Programmieren in C)

Kernideen:

- modelliere Informationen mit Variablen und Daten
- beschreibe Systemzustände mit Hilfe von Variablen
- zerlege den gesamten Verarbeitungsprozess in einzelne Schritte
- fasse ggf. mehrere einzelne Schritte zu Funktionen / Prozeduren (Teilprogramme) zusammen
- Konzepte der imperativen Programmierung entstanden als Abstraktion der Arbeitsweise von Rechnern



Bemerkung:

- Die imperative Programmierung baut auf dem klassischen Algorithmusbegriff auf.
- Eine Berechnung wird als zustandsverändernder Ablauf angesehen.

1. Algorithmen, Programme und Software

1. Vom System zum Programm
2. Vom Algorithmus zum Programm
3. Programmierparadigmen
4. **Algorithmische Grundkonzepte**
5. Strukturiertes Programmieren
6. Zusammenfassung & Ausblick

Variablen

Um die Zustände zwischen den Schritten besser fassen zu können, führen wir **Variablen** ein, die Werte speichern können.

- grafische Darstellung: durch Rechtecke

Beispiele:

- v: `true` → Die Variable v enthält/speichert den Wert `true`.
- v_1: `42` → Die Variable v_1 enthält/speichert den Wert `42`.

Definition: Eine **Variable** (auch Speichervariable genannt) ist ein Speicher (Behälter) für Werte. Typische Operationen auf einer Variablen v sind

- das Zuweisen eines Wertes w an v
- das Auslesen des Wertes, den v gespeichert hat.

Bemerkung:

- Der Zustand einer Variablen v ist undefiniert, wenn man ihr noch keinen Wert zugewiesen hat.
- Anderenfalls ist der Zustand von v durch den in ihr gespeicherten Wert gegeben.

Detailentwurf mit Pseudocode

Pseudocode ist eine künstliche, informelle Sprache zur textuellen Formulierung von Algorithmen

- Aktionen und Reihenfolge
- Einsatz zum Detailentwurf
- Pseudocode-“Programme“ sind nicht auf einem Rechner ausführbar

Beispiel:

```
erzeuge Variablen x, y und sum
fordere auf zur Eingabe von Werten für x und y
berechne sum als Summe x+y
gib sum aus
```

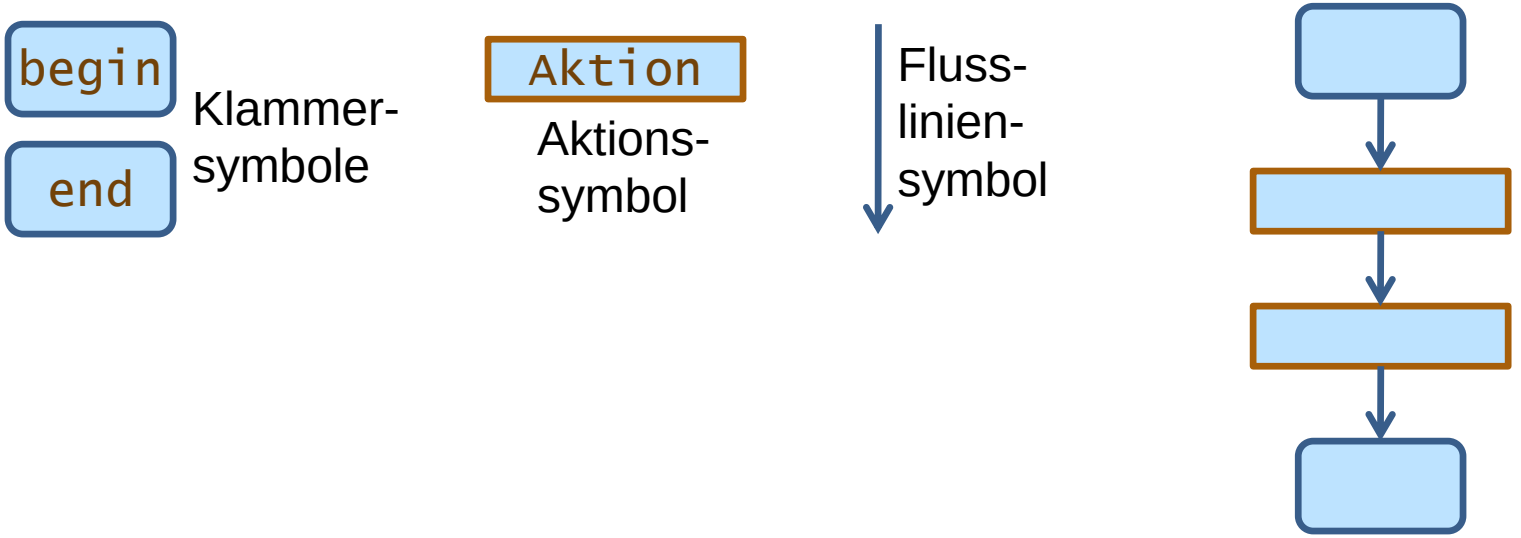
Grafische Alternativen zum Pseudocode (1)

Ein **Flussdiagramm** ist eine grafische Repräsentation eines Algorithmus.

- Einsatz zum Detailentwurf
- Alternative zum Pseudocode

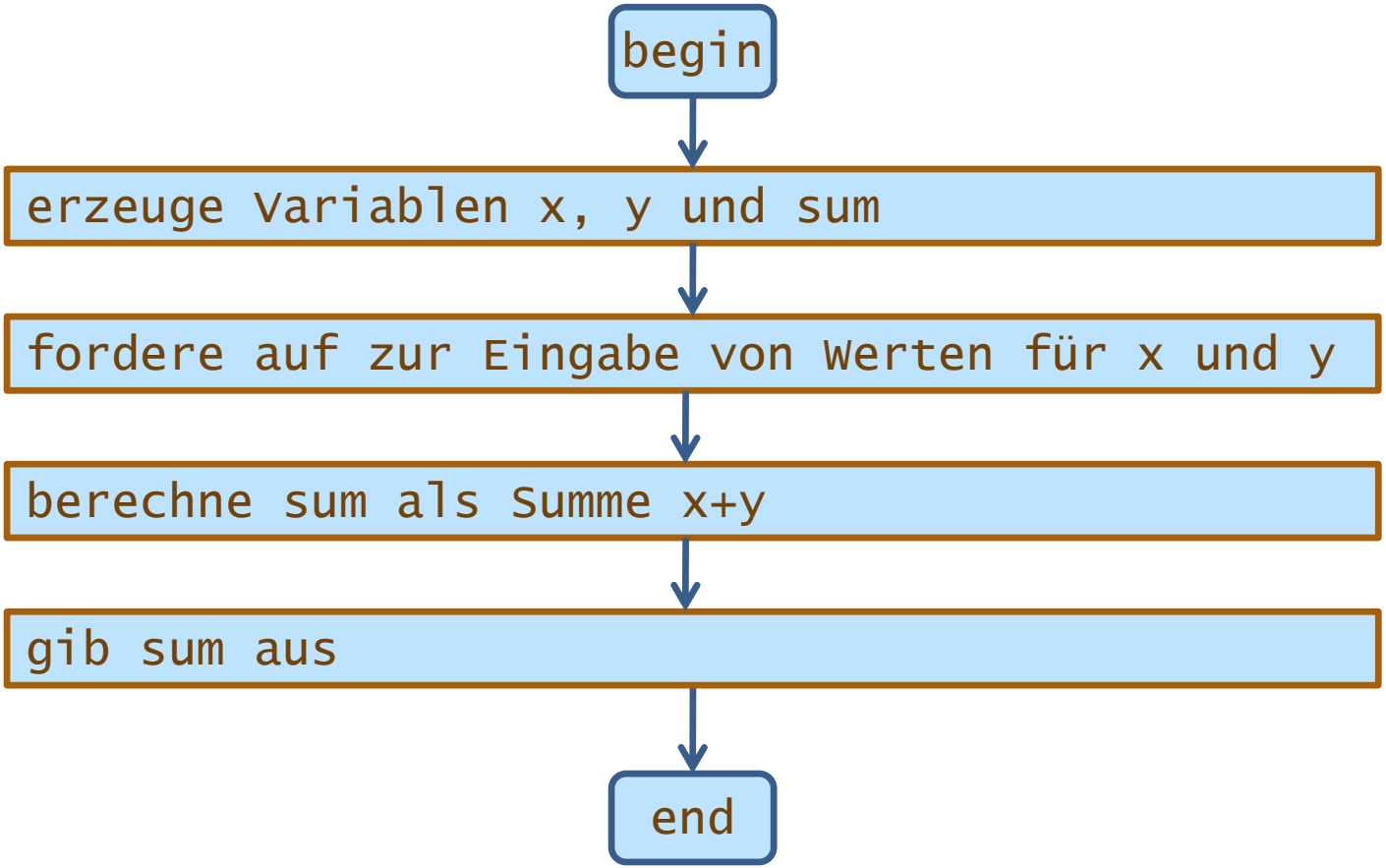
Nutzung spezieller grafischer Symbole zur informellen Beschreibung von Aktionen

Verbindung mit Flusslinien zur Festlegung des Kontrollflusses



Flussdiagramm: Beispiel

Detailentwurf des Additionsprogramms



Grafische Alternativen zum Pseudocode (2)

Struktogramme sind eine alternative grafische Darstellung für Programmentwürfe.

- auch bekannt als Nassi-Shneiderman-Diagramme

grafische Repräsentationen für

- Blöcke von Anweisungen
- Verzweigungen und Fallauswahlen
- Wiederholungen (Schleifen)

fordern und ermöglichen „mehr Struktur“ als Flussdiagramme

Ein weiterer Algorithmus (1)

Beispiel: Berechnung der Fakultät (n!)

- umgangssprachliche Formulierung in Pseudocode

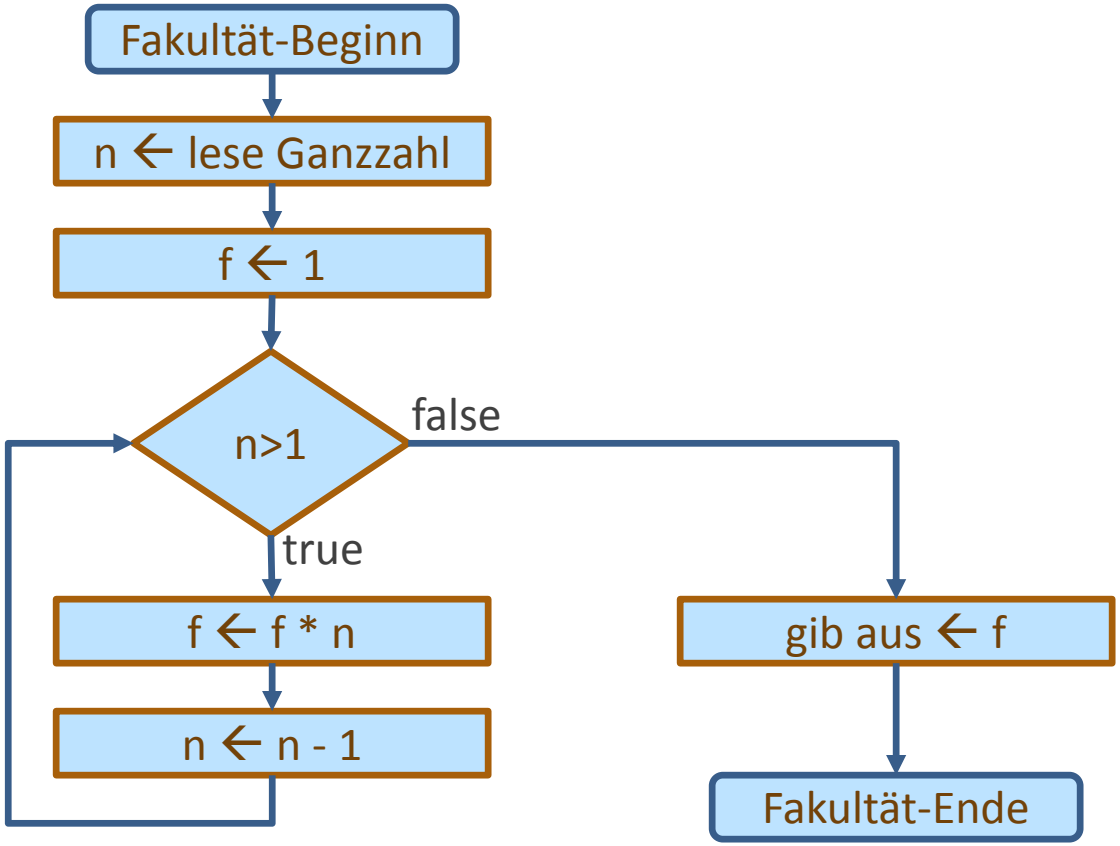
```
BERECHNUNG DER FAKULTÄT:

seien n und f Variablen für ganze Zahlen

lese durch Benutzereingabe den Wert n
  ein, für den die Fakultät berechnet
  werden soll

weise 1 an f zu

solange der Wert von n größer als 1 ist,
  tue Folgendes und prüfe danach wieder
  die Bedingung:
  1. multipliziere f mit n und weise das
     Ergebnis an f zu
  2. reduziere den Wert von n um 1

gebe den Wert aus, den f enthält
```

Exemplarischer Ablauf:

	n	f
Eingabe	4	
Zuweisung		1
n=4 > 1		4 (1.)
	3 (2.)	
n=3 > 1		12 (1.)
	2 (2.)	
n=2 > 1		24 (1.)
	1 (2.)	
n=1 ≤ 1	Schleifenabbruch, Ausgabe von 24 (Wert von f) → Fakultät von 4 ist 24.	

Ein weiterer Algorithmus (2)

Alternative Formulierung des Algorithmus durch ein **Flussdiagramm**



Ein weiterer Algorithmus (3)

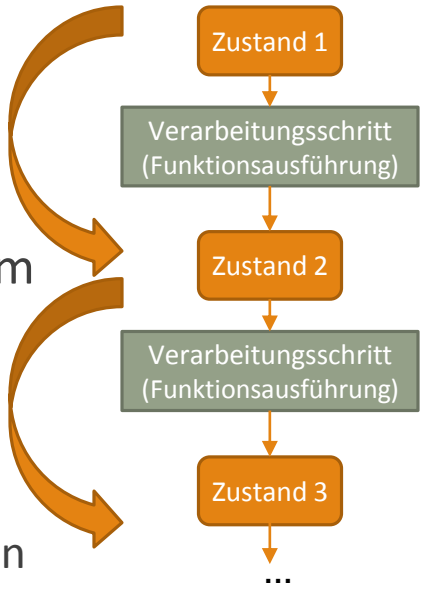
Alternative Formulierung des Algorithmus durch ein **C-Programm**

```
#include <stdio.h>
void main()
{
  int n, f;
  scanf( "%d", &n );
  f = 1;
  while ( n > 1 )
  {
    f = f * n;
    n = n - 1;
  }
  printf( "%d", f );
}
```

Bemerkungen:

- Algorithmen können mehr oder weniger formal beschrieben sein.
- Ein Algorithmus ist unabhängig von der verwendeten Beschreibungstechnik bzw. Programmiersprache.

Zustände



Jeder Schritt bei der Ausführung eines Algorithmus führt von einem Ausführungszustand (kurz: Zustand) zu einem Nachfolgezustand.

Definition: Ein **Ausführungszustand** ist gekennzeichnet durch

- den **Speicherzustand** (i.W. die Zustände sämtlicher Variablen)
- den **Steuerungszustand** (vereinfacht gesagt, die Stelle im Programm, an der die Ausführung gerade angekommen ist)

Ausführungszustand des Algorithmus

Welche Werte haben alle Variablen? | *In welcher Zeile ist man gerade?*

Wichtig: Jeder einzelne Ausführungsschritt führt zu einer Zustandsveränderung, also einer Veränderung von Speicherzustand **und/oder** Steuerungszustand.

Beispiel:

- tabellarische Darstellung der Anwendung des Fakultätsalgorithmus

	n	f
Eingabe	4	-
Zuweisung	4	1
n=4 > 1	4	4 (1.)
	3 (2.)	4
n=3 > 1	3	12 (1.)
	2 (2.)	12
n=2 > 1	2	24 (1.)
	...	

Aktionen und Ablauf

In jedem (Ausführungs-)**Schritt** des Algorithmus wird eine **Aktion** ausgeführt.

Aktionen können sein:

- Zuweisungen an Variablen
- Beeinflussung des weiteren Ablauf des Algorithmus (des nächsten Steuerungszustands)
- Kommunikation mit der Umgebung (Ein-/Ausgabe)

Durch die Aktion werden bestimmt

- der nachfolgende Speicherzustand
- der nachfolgende Steuerungszustand
- ggf. die Terminierung des gesamten Algorithmus

Definition: Unter dem **Ablauf** des Algorithmus verstehen wir die vollständige

- Sequenz der Ausführungszustände und
- Sequenz der ausgeführten Aktionen,

die sich zu gegebenen Eingaben ergeben haben.

Eigenschaften von Algorithmen (1)

Effizienz

- Ein Algorithmus A heißt effizienter als ein Algorithmus B, wenn der „Aufwand“ zur Ausführung von A geringer ist als der Aufwand zur Ausführung von B; und zwar für alle zulässigen Eingabedaten.
- In der Praxis wird oft nur erwartet, dass der Aufwand
  - für alle bis auf endlich viele Eingaben (Ausnahmefälle) geringer ist
  - oder aber im Mittel geringer ist über alle zulässigen Eingaben.

Bei den Überlegungen zur Effizienz unterscheidet man zwischen:

- Zeitkomplexität:
  Wie viele Schritte braucht mein Algorithmus in Abhängigkeit der Eingabewerte?
- Speicherkomplexität (Raumkomplexität):
  Wie viel Speicherplatz braucht mein Algorithmus in Abhängigkeit der Eingabewerte?

Iterativität

- Ein Algorithmus heißt iterativ, wenn er Schleifen enthält, so dass Teile von ihm mehrfach durchlaufen werden.

Eigenschaften von Algorithmen (2)

Determinismus

- Ein Algorithmus heißt deterministisch, wenn für alle Eingabedaten der Ablauf des Algorithmus eindeutig bestimmt ist.
  (*Erinnerung:* Ablauf = Sequenz der Ausführungszustände + Sequenz der ausgeführten Aktionen)
- Anders gesagt: Bei wiederholten Ausführungen des Algorithmus für die gleichen Eingabedaten ergibt sich der gleiche Ablauf.
- Anderenfalls heißt der Algorithmus nichtdeterministisch.

Beispiel:

- für einen nichtdeterministischen Algorithmus
- Frage: Was tut der Algorithmus?
- Frage: Warum ist er nichtdeterministisch?

```
seien m, s, w Variablen für ganze Zahlen

lese den Wert n ein (Benutzereingabe)

weise m den Wert 0 zu
weise s den Wert 0 zu

solange der Wert von m kleiner als n
  ist, tue Folgendes und prüfe danach
  wieder die Bedingung:
  1. weise w einen zufälligen Wert aus
     dem Bereich [1,6] zu (Würfeln)
  2. erhöhe den Wert von s um w
  3. erhöhe den Wert von m um 1

gib den Wert aus, den s enthält
```

Eigenschaften von Algorithmen (3)

Determiniertheit

- Ein Algorithmus heißt determiniert, wenn er bei gleichen zulässigen Eingabewerten stets das gleiche Ergebnis liefert.
- Anderenfalls heißt er nichtdeterminiert.
- Beispiele:
  - Jeder Algorithmus, der eine (mathematische) Funktion berechnet, ist determiniert.
  - Das Beispiel auf der vorigen Folie ist nichtdeterminiert.
- Übungsaufgabe: Konstruieren Sie einen Algorithmus, der determiniert, aber nichtdeterministisch ist.

Terminierung

- Die Ausführung eines Algorithmus terminiert, wenn sie nach endlich vielen Schritten beendet ist.
- Anderenfalls spricht man von einer nichtterminierenden Ausführung.
- Ein Algorithmus heißt dementsprechend terminierend, wenn garantiert ist, dass er terminiert.

1. Algorithmen, Programme und Software

1. Vom System zum Programm
2. Vom Algorithmus zum Programm
3. Programmierparadigmen
4. Algorithmische Grundkonzepte
5. **Strukturiertes Programmieren**
6. Zusammenfassung & Ausblick

Strukturierte Programmierung

Die Reihenfolge der Ausführung von Aktionen wird unter Verwendung von **Kontrollstrukturen** festgelegt.

Es lässt sich mathematisch beweisen, dass alle Algorithmen unter Verwendung von lediglich drei Kontrollstrukturen formulierbar sind:

Anweisungen

Anweisungen sind dazu da, um Kontrollstrukturen umzusetzen.

Anweisungen sind programmiersprachliche Beschreibungsmittel.

- Einfache Anweisungen beschreiben Aktionen.
- Zusammengesetzte Anweisungen beschreiben, wie mehrere Anweisungen in Folge auszuführen sind.
- Zusammengesetzte Anweisungen bilden Teile von Algorithmen.

Anweisungen: Überblick

Arten von Anweisungen und ihre Realisierung (in C)

1. Algorithmen, Programme und Software

1. Vom System zum Programm
2. Vom Algorithmus zum Programm
3. Programmierparadigmen
4. Algorithmische Grundkonzepte
5. Strukturiertes Programmieren
6. **Zusammenfassung & Ausblick**

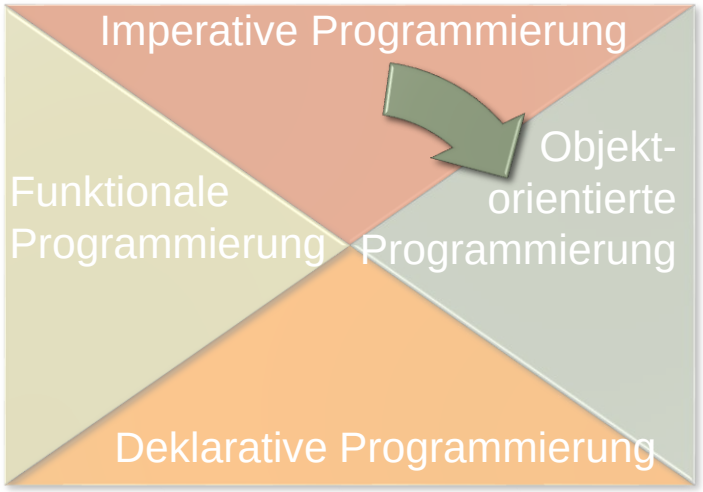
Ausblick: Weitere Aspekte der strukturierten Programmierung

Strukturierte Programmierung

- Kontrollstrukturen
- Anweisungen
- Variablen in Programm und Speicher

Funktionen (Prozeduren, Unterprogramme)

- Rekursion
- Benutzerdefinierte Typen
- Sichtbarkeit von Bindungen
- Iteration
- Parameterübergabe
- Seiteneffekte



Zusammenfassung (1)

```
Thu ihm also:
Schreib die zahl vor dich
mach ein Linien darunter
heb an zu forderst
Duplir die erste Figur.
Kompt ein zahl die du mit einer Figur /
  schreiben magst
so setz die unden.
Wo mit zweyen
schreib die erste
Die ander behalt im sinn.
Darnach duplir die ander
und gib darzu
das du behalten hast
und schreib abermals die erste Figur
wo zwo vorhanden
und duplir fort bis zur letzten
die schreibe ganz aus
als folgende Exempel ausweisen.
```

„Dupliren“
nach Adam Riese (1574):
„Lehret wie du ein zahl zweyfaltigen solt.“

- nicht-abweisende Schleife
- Fallunterscheidung
- Anweisungsblock

Zusammenfassung (2)

Programmiersprachen und -paradigmen
- imperative Programmierung

Strukturierte Programmierung
- 3 Arten von Kontrollstrukturen ausreichend

Realisierung der Kontrollstrukturen
- Pseudocode & Flussdiagramme
- Umsetzung in C → weitere Vorlesungen