Informatik 1

Einführung in die Informatik und Programmierung

Dozent: Jonas Fritzsch, M.Sc.

Kontakt: fritzsch@lehre.dhbw-stuttgart.de

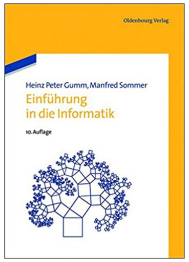
Organisatorisches

- Vorlesungszeiten, Termine
- Übungen, Labor
- Webseite zur Vorlesung
  http://wwwlehre.dhbw-stuttgart.de/~fritzsch/Informatik/
- Skript
- Klausur

Inhalte der Vorlesung

- Grundlegende Konzepte der Programmierung
- Algorithmisches Denken schulen
  Problem → Algorithmus → Programm
- Programmiersprache C
- Grundlegende Algorithmen
  (Suchen, Sortieren, Listen, Felder, …)
- üben, üben, üben, …

Literatur



Einführung in die Informatik / Gumm, Sommer /
De Gruyter Oldenbourg, 2012

Grundlagen der Informatik / Herold et al.
Pearson Studium, 2017

Programmieren in C. ANSI C (2. Ausgabe) / Brian W. Kernighan,
Dennis M. Ritchie / Hanser Fachbuch

C von A bis Z / Wolf / Galileo Computing
http://openbook.galileocomputing.de/c_von_a_bis_z/

C als erste Programmiersprache / Manfred Dausmann et al. /
Vieweg+Teubner Verlag

C Programmieren von Anfang an / Helmut Erlenkötter /
rororo Computer

Einführung in die Informatik

Gliederung

Allgemeines

Digitalrechner

Algorithmen

Sprachen

Allgemeines

Was ist Informatik? (Information – Automatik, Mathematik)

Def. 1 Informatik ist die Wissenschaft von der **systematischen Verarbeitung von Informationen** - insbesondere deren maschinelle (automatische) Bearbeitung, Speicherung und Übertragung durch Digitalrechner (Computer). *(Informatik Duden)*

Def. 2 Informatik umfasst die automatisierte Informationsverarbeitung in Natur, Technik und Gesellschaft *(Fachliteratur "Grundlagen d. Informatik")*

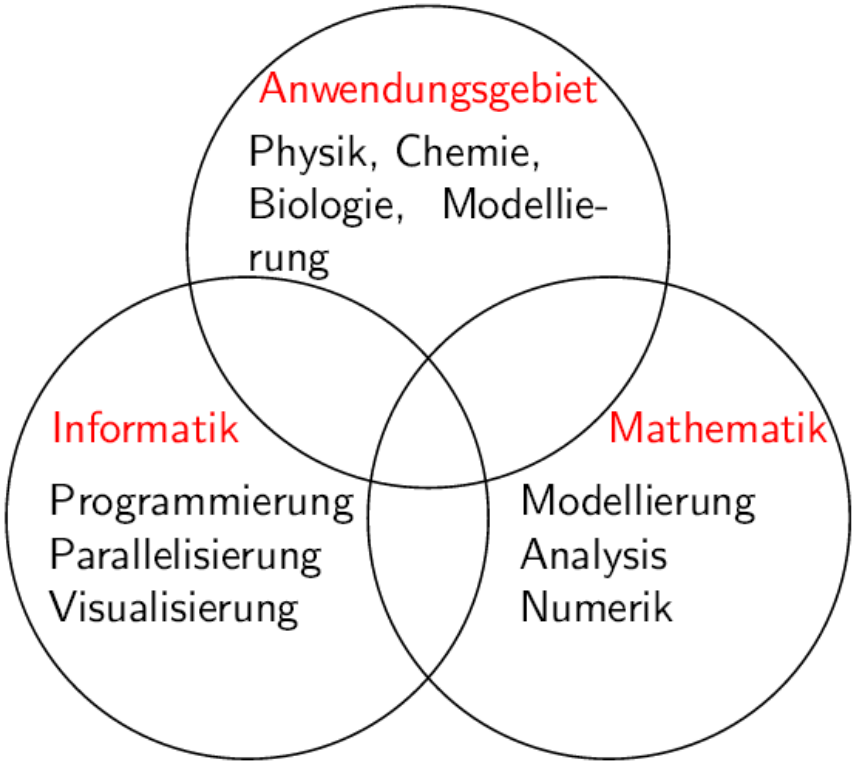
- Begriff 1957 von Karl Steinbuch eingeführt
- vgl. "computer science"

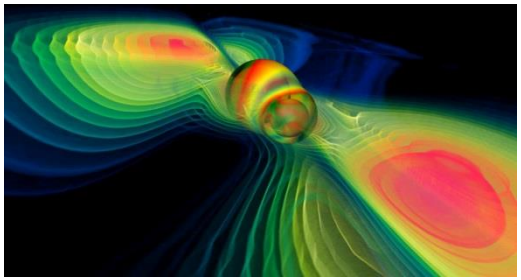
  E. W. Dijkstra: „Computer Science is no more about computers than astronomy is about telescopes“

  → Informatik vs. Rechnertechnik
- Ende der 1960er Jahre erstmals Studienrichtung

Interdisziplinäre Forschungsdisziplin

Beispiel: Wissenschaftliches Rechnen in Experimenten

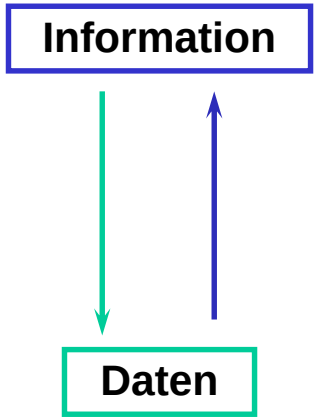




Teilgebiete der Informatik

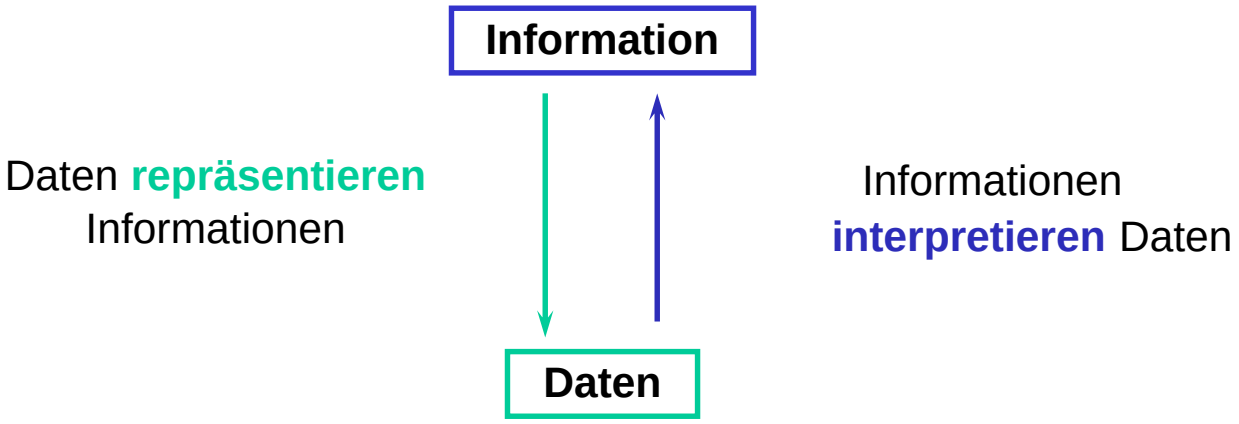
Theoretische Informatik	Praktische Informatik	Angewandte Informatik	Technische Informatik
Sprachen u. Automaten	Programmier-sprachen	Ingenieur-bereich	Rechnertechnik
formale Logik	Compiler Interpreter	Wirtschaft / Verwaltung	Rechner-architektur
Berechen-barkeit	Info- / DB-Systeme	Medizin / Med.Technik	Prozeßdaten-verarbeitung
Komplexitäts-Theorie	Betriebs-systeme	Methoden Entwicklung	Mikroelektronik
Informations-/ Codierungsth.	Künstl. Intelligenz		

Information und Daten

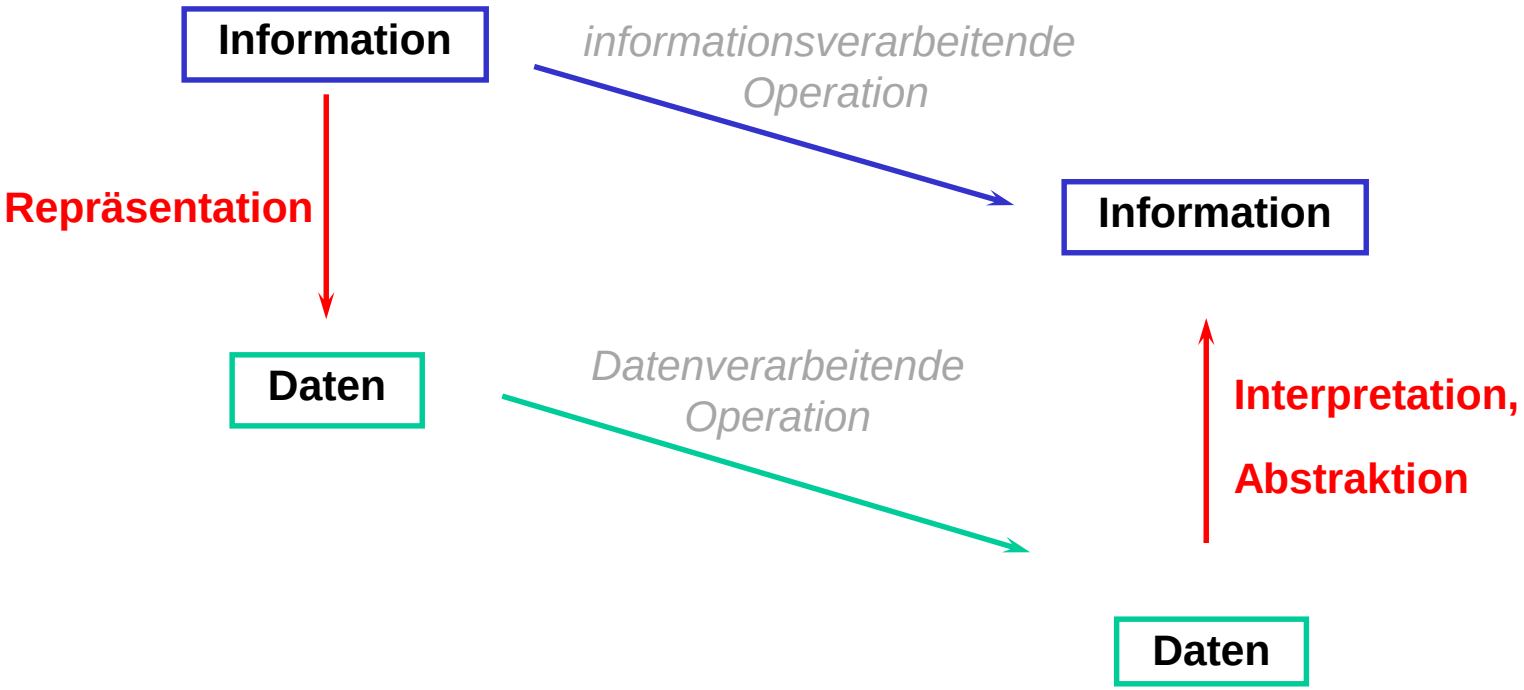


Information und Daten

Information ≙ Bedeutungsgehalt von Zeichen, Nachrichten



Informationsverarbeitung und Datenverarbeitung



Information

- Information (v. lat.: in-form-are) = Kenntnis über irgendwas
- Austausch über **Nachrichten**

Nachricht	Zusammenstellung von Zeichen (Symbolen) zur Informationsübermittlung
Zeichen/Symbol	Element eines Zeichenvorrates festgelegter Menge (z.B. Alphabet)
Wort	Folge von Zeichen, die als Einheit betrachtet werden

Information

- Information (v. lat.: in-form-are) = Kenntnis über irgendwas
- Austausch über **Nachrichten**

Nachricht	Zusammenstellung von Zeichen (Symbolen) zur Informationsübermittlung
Zeichen/Symbol	Element eines Zeichenvorrates festgelegter Menge (z.B. Alphabet)
Wort	Folge von Zeichen, die als Einheit betrachtet werden

Beispiel 1: Bei welcher Nachricht ist die Information größer?

a) Am 1. Juli war die Temperatur größer als 25 Grad.
b) Am 1. Juli betrug die Temperatur 29 Grad.

Information

- Information (v. lat.: in-form-are) = Kenntnis über irgendwas
- Austausch über **Nachrichten**

Nachricht	Zusammenstellung von Zeichen (Symbolen) zur Informationsübermittlung
Zeichen/Symbol	Element eines Zeichenvorrates festgelegter Menge (z.B. Alphabet)
Wort	Folge von Zeichen, die als Einheit betrachtet werden

Beispiel 1: Bei welcher Nachricht ist die Information größer?

a) Am 1. Juli war die Temperatur größer als 25 Grad.
b) Am 1. Juli betrug die Temperatur 29 Grad.

c) Am 1. Juli war die Temperatur größer als 25 Grad.
d) Am 1. Januar war die Temperatur größer als 25 Grad.

Informationsübertragung

Gehalt an Information unterscheiden sich je nach Empfänger
→ **Informationsgehalt einer Nachricht**

Shannonsches Informationsmaß H: $I(x) = log_2 N \ bit$

bei N = Gesamtzahl der verwendeten Zeichen
und gleicher Wahrscheinlichkeit der Zeichen $x_1 \ldots x_n$
mit $p(x_1) = p(x_2) = \ldots p(x_n)$
und $p(x_1) + p(x_2) + \ldots + p(x_n) = 1$

Beispiel 2: Zeichenvorrat: 10 Zeichen {0, 1, 2, 3, 4, 5, 6, 7,8 ,9}, die mit gleicher Wahrscheinlichkeit auftreten.

Informationsgehalt einer Nachricht bestehend aus 1 Zeichen ist somit:

$$I(x) =$$

Informationsübertragung

aus Beispiel 1: Bei welcher Nachricht ist die Information größer?

c) Am 1. Juli war die Temperatur größer als 25 Grad
d) Am 1. Januar war die Temperatur größer als 25 Grad.

Informationsübertragung

aus Beispiel 1: Bei welcher Nachricht ist die Information größer?

c) Am 1. Juli war die Temperatur größer als 25 Grad
d) Am 1. Januar war die Temperatur größer als 25 Grad.

- je seltener ein Zeichen x auftritt, desto größer sein **Informationsgehalt** $\boldsymbol{I(x)}$
  genauer: $I(x) = \sim \frac{1}{p(x)}$
- für den Fall $p(x) = 1$ soll gelten: $I(x) = 0$
- es folgt: $I(x) = \log_2\left(\frac{1}{p(x)}\right)$

Informationsübertragung

aus Beispiel 1: Bei welcher Nachricht ist die Information größer?

c) Am 1. Juli war die Temperatur größer als 25 Grad
d) Am 1. Januar war die Temperatur größer als 25 Grad.

- je seltener ein Zeichen x auftritt, desto größer sein **Informationsgehalt** $I(x)$
  genauer: $I(x) = \sim \frac{1}{p(x)}$
- für den Fall $p(x) = 1$ soll gelten: $I(x) = 0$
- es folgt: $I(x) = \log_2\left(\frac{1}{p(x)}\right)$

Schlussfolgerungen:

Information ≙ Maß der **Ungewissheit**, mit der ein Empfänger eine Nachricht erwarten kann.

1 bit ≙ Information, welche bei **gleicher** Wahrscheinlichkeit zweier Alternativen durch die Kenntnis einer Alternative vermittelt wird.

Bits

Informationen werden repräsentiert als Folgen von Bits. **Bit = B**inary Digi**t**

Verwendung:
- Binärziffer
- Maßeinheit für Datenmenge
- Maßeinheit für Informationsgehalt

- Wert 0 oder 1 / aus oder an / ja oder nein
- elektrische Spannung: 0 = 0 Volt 1 = 5 Volt
- Magnetisierung: 0 = entmagnetisiert 1 = magnetisiert

Bitfolgen

- mehr als 2 Zustände abbilden
- Bitfolge aus 2 Bits = 4 Zustände

  00 01 10 11

  → es gibt genau 2^N mögliche Bitfolgen der Länge N Bit

Beispiel: Windrichtung

Frage: Aus welcher Himmelsrichtung weht der Wind?

Codierung von Windrichtungen:

Nord
Süd
Ost
West
Nordwest
Nordost
Südwest
Südost

Hexziffern

- Folgen von Nullen/Einsen sind unübersichtlich

```
01001111011000010110110001101100
```

- Idee: Anordnung in Gruppen zu 4 Bits

```
0100 1111 0110 0001 0110 1100 0110 1100
```

- **Halb-Byte:** 16 Zustände
- Ziffern '0' bis '9' und Buchstaben 'A' bis 'F'

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

Bytes

- Oktett von Bits: **8 Bits = 1 Byte**

- Beispiel: $(11000101)_{2} = (C5)_{16} = (197)_{10}$

Beispiele der Codierung:
- Zahl zwischen 0 und 255
- Zahl zwischen -128 und +127
- Zeichen im Zeichencode, z.B. ASCII Code

Einheiten:

1 Kilobyte	2^{10}	1024 Bytes
1 Megabyte	2^{20}	1.048.576 Bytes
1 Gigabyte	2^{30}	1.073.741.824 Bytes
1 Terabyte	2^{40}	1.099.511.627.776 Bytes
1 Petabyte	2^{50}	1.125.899.906.842.624 Bytes

ASCII Code

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- Abbildungsvorschrift zur binären Codierung von Zeichen

- 7 Bit = 128 Zeichen

- 33 nicht-druckbare, 95 druckbare Zeichen

- erweiterte Codes für weitere 128 Zeichen (sprachspezifisch)

Letter	Ascii Code	Hex Number	Binary
H	72	48	1001000
e	101	65	1100101
l	108	6C	1101100
l	108	6C	1101100
o	111	6F	1101111
	32	20	100000
E	69	45	1000101
l	108	6C	1101100
l	108	6C	1101100
i	105	69	1101001
e	101	65	1100101
:	58	3A	111010
	32	20	100000
H	72	48	1001000
o	111	6F	1101111
w	119	77	1110111
	32	20	100000
a	97	61	1100001
r	114	72	1110010
e	101	65	1100101
	32	20	100000
y	121	79	1111001
o	111	6F	1101111
u	117	75	1110101
	32	20	100000
t	116	74	1110100
o	111	6F	1101111
d	100	64	1100100
a	97	61	1100001
y	121	79	1111001
.	46	2E	101110

Digitalrechner

Wurzeln der Rechnertechnik

1100 v.Chr. Abakus

16.Jh. Adam Ries(e): Rechengesetze zum Dezimalsystem

1623 Wilhelm Schickard: Erste Rechenmaschine für Kepler

17.Jh. weitere Rechenmaschinen von B. Pascal und G. Leibnitz

1774 Philipp Hahn: erste zuverlässige mechanische Rechenmaschine

1838 Charles Babbage (Eng): "Analytical Engine", Differenzmaschine (kann polynomiale Funktionen auswerten) mechanischer Vorläufer heutiger programmierbarer Rechner

1880 Hermann Hollerith (USA): Auswertung der Volkszählung in 4 Wochen statt 7 Jahren, Erfinder der Lochkarten

Wurzeln der Rechnertechnik

1938 Zuse: Z1 **elektromechanischer** Rechenapparat, Binärsystem, Lochstreifen-Programmsteuerung

1940 Zuse: Z2 verbesserte Version mit Telefon-Relais

1941 Zuse: Z3 erster funktionsfähiger programmgesteuerter Rechenautomat mit 2600 Relais, 30-60 Ops/s

1949 Zuse: Z4 erster kommerziell verfügbarer Computer

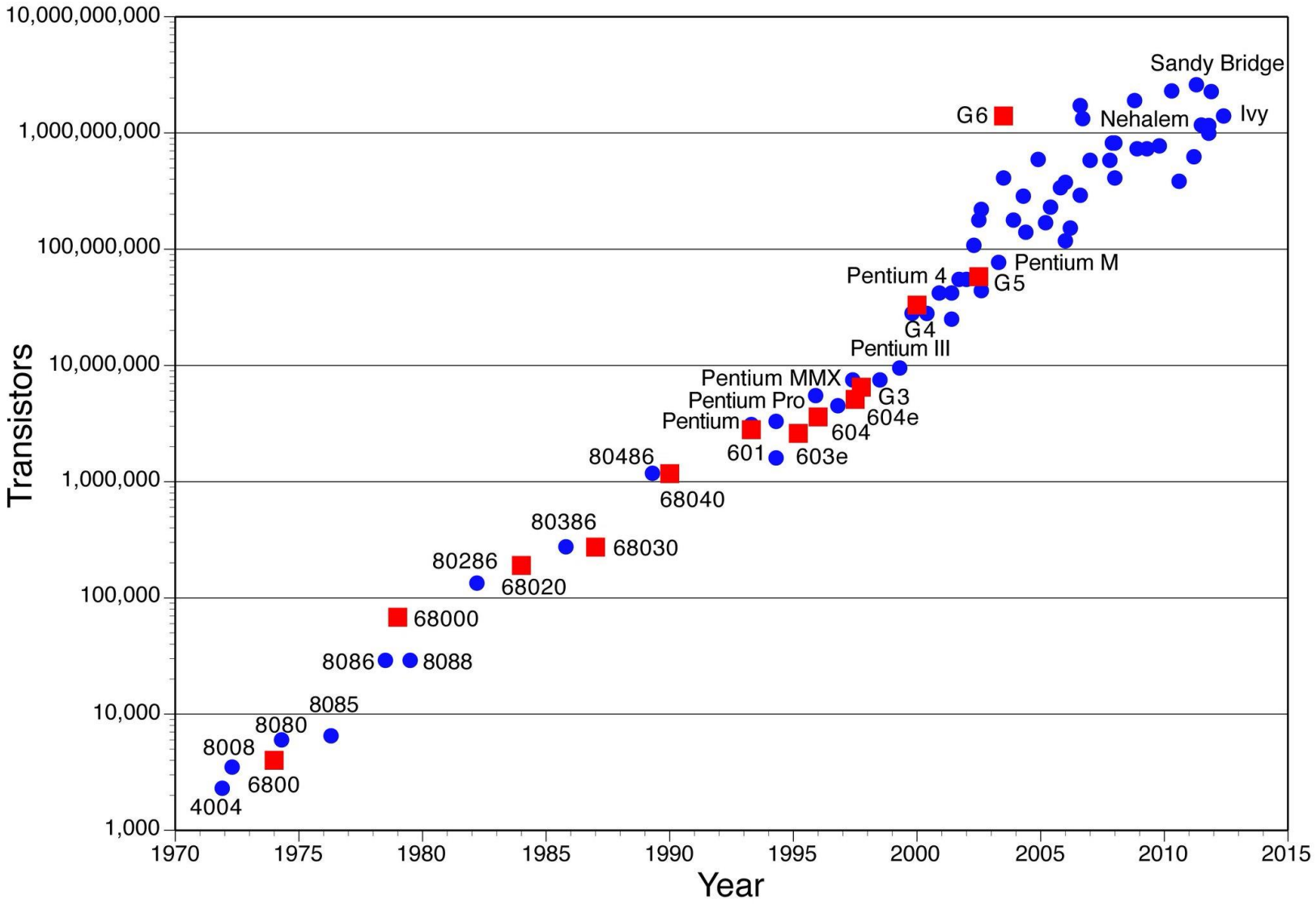
1944 Howard G. Aiken (USA) / Harvard University: MARK 1 Lochkartenrechner

1946 1. Generation: Elektronische Röhrenrechner (ENIAC, USA, 17000 Röhren, 30 Tonnen, 140 qm, 174kW, Geschwindigkeit 100-mal höher als Mark I)

1958 2. Generation: Transistor-Rechner

1964 3. Generation: Integrierte Chips (LSI, Large Scale Integration)

1970 4. Generation: VLSI-Technik: Zusammenfassen vieler Chips zu einem
1980 hochintegrierten Baustein (MOS Technologie)


10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
Transistors
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
Year
Sandy Bridge
Ivy
Nehalem
G6
Pentium M
Pentium 4
G5
G4
Pentium III
Pentium MMX
Pentium Pro
G3
604e
Pentium
604
601
603e
80486
68040
80386
68030
80286
68020
68000
8086
8088
8085
8080
8008
6800
4004

Computeranwendung heute

- **Kommerzielle Rechner / Internet / Mobile Endgeräte**

  Ein-/Ausgabe großer Datenmengen, aber eher einfache Berechnungen

- **Wissenschaftliche Rechner**

  komplexe, langwierige Rechnungen, aber eher kleine Mengen von Ein-/Ausgaben

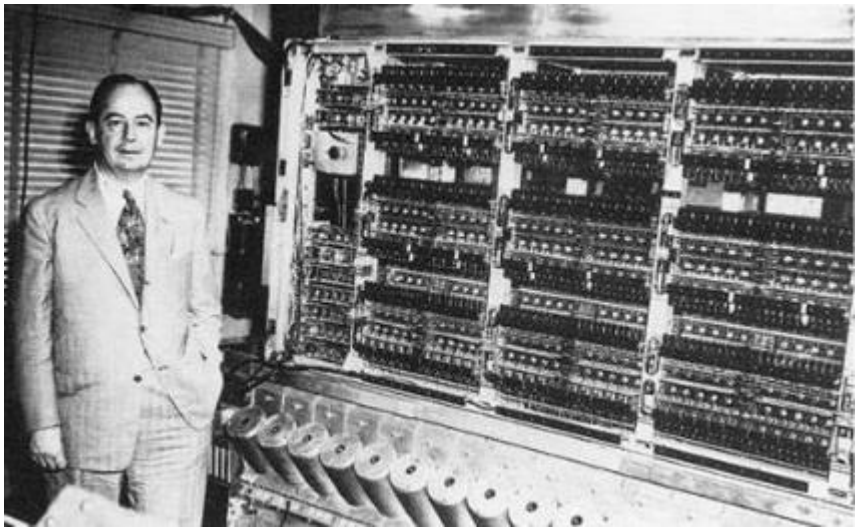
- **Prozess-/Echtzeit-Rechner**

  Steuerung oder Überwachung von technischen Prozessen.

  neben logischer Korrektheit des Ergebnisses ist die ”zeitliche Korrektheit“ ebenso wichtig

Hardware: Die Maschine

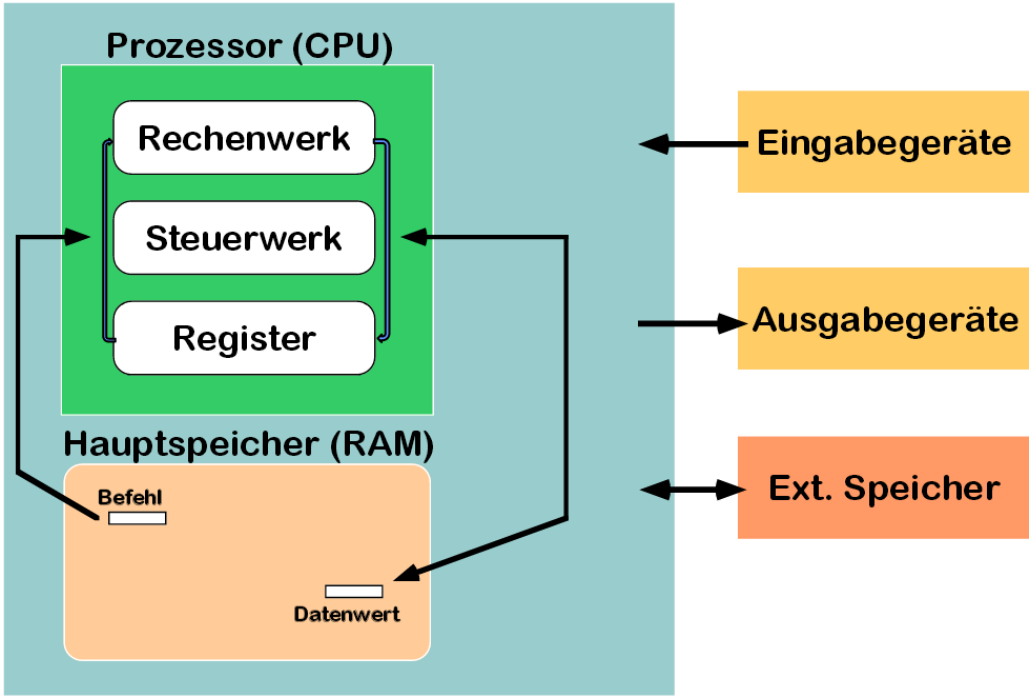
- John **Von-Neumann-Modell** (1947)
- EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe)
- allzwecktauglich (General Purpose Computer)
- nicht selbstständig arbeitsfähig, Anwendungsmaschine durch Software
- Einfachheit (übersichtlich, minimaler HW-Aufwand)
- maximale Flexibilität (bei genügend elementaren Befehlen)



Registermaschine auf Basis des von-Neumann-Modells

- Befehle und Daten im gleichen Speicher
- sequentielle Verarbeitung von Befehl und Datum
- einzelner Verbindungsweg zwischen CPU ↔ Speicher

(zwischen CPU und Speicher wird immer nur ein Wort transportiert)



Beispiel: Programm "Summe"

Aufgabe: Summe von einzugebenden Zahlen ermitteln und ausgeben,
Abbruch bei Eingabe von "0"

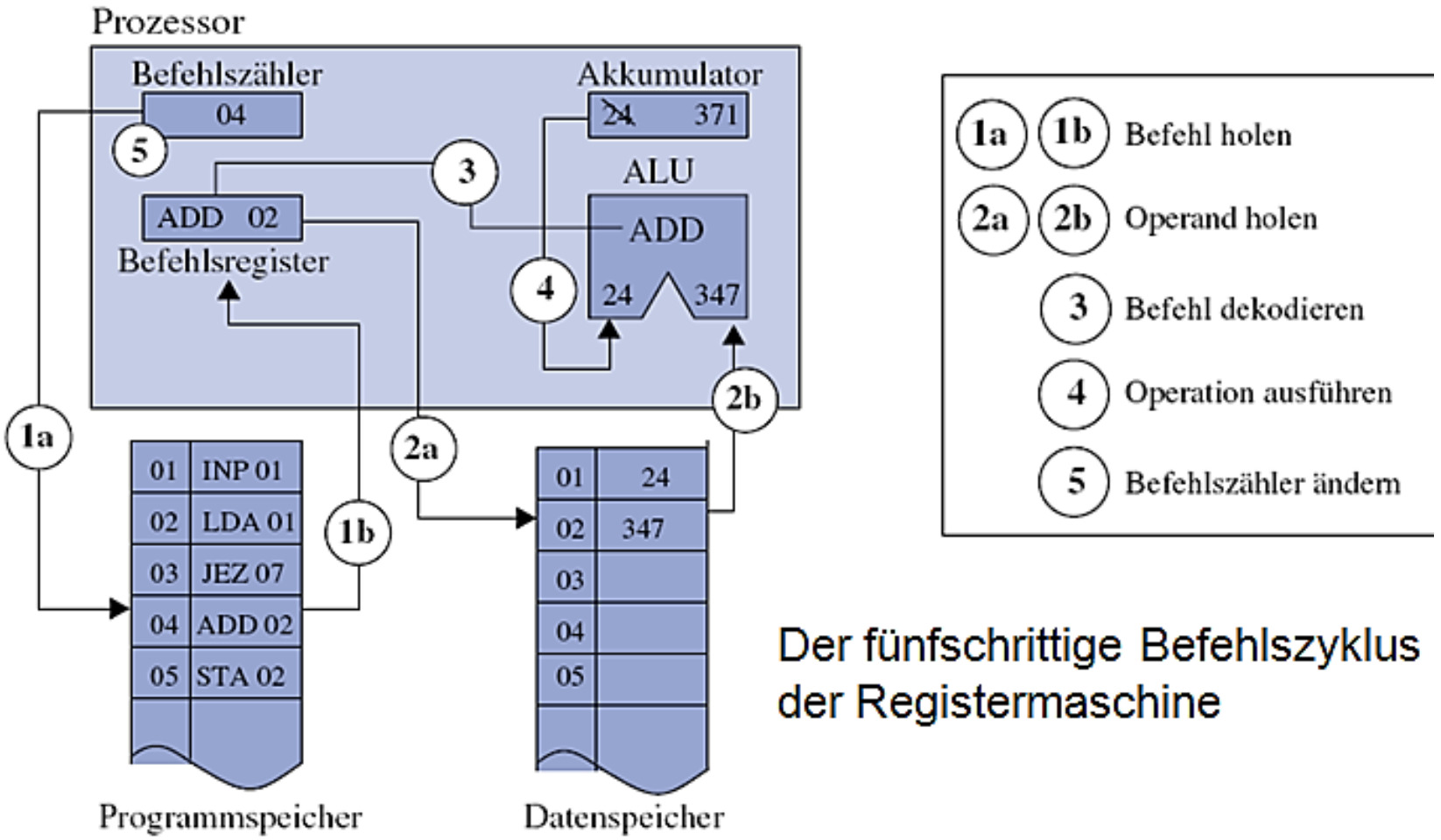
Programmstart
  Zahl einlesen
  Wenn Zahl = 0, springe zur Ergebnisausgabe
  Zahl auf bisherige Summe aufaddieren
  Springe zum Programmstart
  Ergebnis ausgeben
Programmende

Beispiel: Programm "Summe"

Aufgabe: Summe von einzugebenden Zahlen ermitteln und ausgeben,
Abbruch bei Eingabe von "0"

Programmstart
  Zahl einlesen
  Wenn Zahl = 0, springe zur Ergebnisausgabe
  Zahl auf bisherige Summe aufaddieren
  Springe zum Programmstart
  Ergebnis ausgeben
Programmende

```
01  INP  01     ; Zahl einlesen und an Adr. 1 (Datenspeicher) speichern
02  LDA 01      ; Lade Zahl von Adr. 1 (Datenspeicher) in Akkumulator
03  JEZ  07     ; Falls  Akku == 0, springe an Programmadr. 7 (OUT 02)
04  ADD 02      ; Addiere auf Akku Inhalt von Adresse 2 (Datenspeicher)
05  STA 02      ; Speichere Akku an Adresse 2 (Datenspeicher)
06  JMP 01      ; Springe zurück an Programmadr. 1 (INP 01)
07  OUT 02      ; Gib Inhalt von Adresse 2 (Datenspeicher) aus
08  HLT 99      ; Programmende
```



Der fünfschrittige Befehlszyklus der Registermaschine

Von-Neumann-Rechner/Assembler Simulatoren

Von-Neuman-CPU-Simulator
http://www.ruhr-uni-bochum.de/lpi/cpu/Help.html

Johnny - A Simulator of a Simple von-Neumann Computer
http://sourceforge.net/projects/johnnysimulator/

AssemberSim
http://www.assemblersim.de/

GNUSim8085
http://www.gnusim8085.org

Programm

- Folge von Maschinenbefehlen für einen bestimmten Rechner
- in Programmiersprache formulierter **Algorithmus**

Definition nach Wirth (Nicolaus Wirth, Erfinder der Sprache Pascal):

Programm = Datenstrukturen und Algorithmen

Ein Programm beschreibt somit:
- den Ablauf eines/mehrerer Algorithmen
- die dabei verwendeten Daten und Datenstrukturen
- die Operationen zur Manipulation dieser Daten

Prozeß

- tatsächliche Ausführung eines Programms
- auf maschinellen (oder auch menschlichen) Prozessoren

Algorithmen

Algorithmus = Lösungsverfahren

Menge von Regeln für ein **Verfahren**, um aus gewissen **Eingabegrößen** bestimmte **Ausgabegrößen** herzuleiten

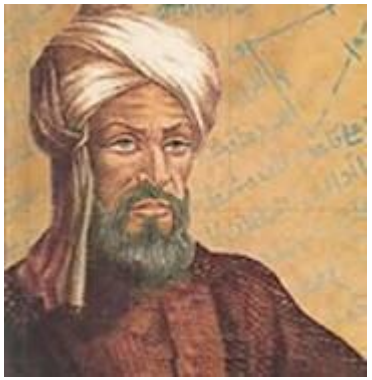
"Der Algorithmus ist als **Oberbegriff zu Programm** zu betrachten, bei dem im Gegensatz zu einem Programm die strengen syntaktischen Regeln der Programmiersprache nicht eingehalten zu werden brauchen"

aus "Programmieren in C" R.Klima/S.Selberherr

"Wenn ein Algorithmus für ein Problem existiert, dann ist das Problem durch Computer lösbar."

"Dixit algorismi" (= "so sprach Al Khowarismi")

Al Khowarismi (783-850): "Regeln der Wiedereinsetzung und Reduktion"



Beispiel: "Gulaschsuppe zubereiten"

Zu manipulierende Objekte:

350 g Rindfleisch, 3 Zwiebeln, 50 g Schweineschmalz, 15 g Mehl, 1 kleine Dose Tomatenmark, 3/4 l Wasser, Salz, Paprika, Majoran

Hilfsobjekte:

Herd, Kochgeschirr

Anweisungen:

Fleisch und Zwiebeln würfeln
in Schmalz andünsten
mit Mehl bestäuben und kurz anrösten
Tomatenmark zugeben
mit Wasser auffüllen
garen
mit Salz, Paprika und Majoran abschmecken

Eigenschaften

- **Eingabespezifikation (Anforderungen)**
- **Ausgabespezifikation (Zusicherungen)**
- **Finitheit**
- **Effektivität**
- **Determiniertheit**
- **Determinismus**
- **Abstrahierung**
- **Terminierung**

Bekannte und Typische Algorithmen

- Euklidischer Algorithmus zur Bestimmung des ggT
- Algorithmen zur Bestimmung von Primzahlen
- Gaußscher Algorithmus zur Lösung eines LGS
- Suchen und Sortieren
- Bearbeiten bestimmter Datenstrukturen (z.B. Listen, Bäume, Graphen)
- Zeichenketten-Verarbeitung
- Mustererkennung
- Statistische Berechnungen

Collatz-Problem

Bildungsgesetz:

- Beginne mit irgendeiner natürlichen Zahl $n > 0$
- Ist n gerade, so nimm als nächstes $n/2$
- Ist n ungerade, so nimm als nächstes $3n + 1$
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl

Collatz-Problem

Bildungsgesetz:

- Beginne mit irgendeiner natürlichen Zahl $n > 0$
- Ist n gerade, so nimm als nächstes $n/2$
- Ist n ungerade, so nimm als nächstes $3n + 1$
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl

So erhält man zum Beispiel für die Startzahl 19 die Folge:

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, …

Collatz-Problem

Bildungsgesetz:

- Beginne mit irgendeiner natürlichen Zahl $n > 0$
- Ist n gerade, so nimm als nächstes $n/2$
- Ist n ungerade, so nimm als nächstes $3n + 1$
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl

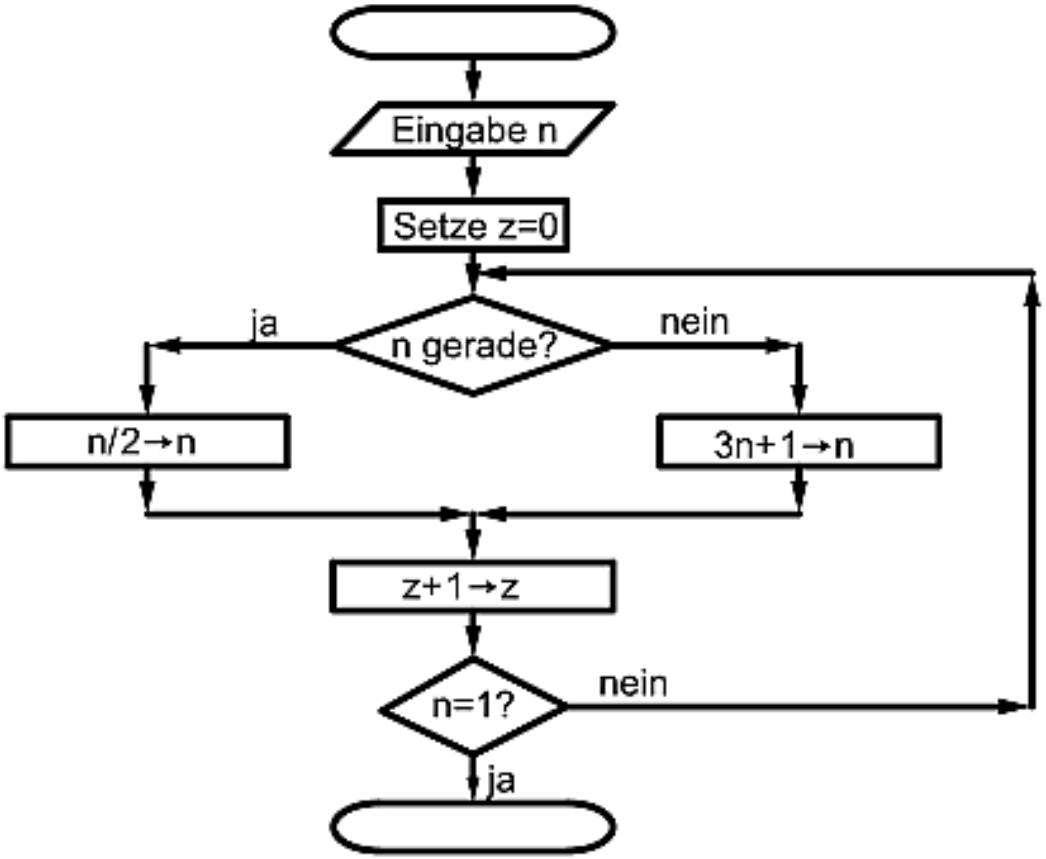
So erhält man zum Beispiel für die Startzahl 19 die Folge:

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, …

Collatz-Vermutung:

Jede so konstruierte Zahlenfolge mündet in den Zyklus 4, 2, 1,
egal, mit welcher natürlichen Zahl n man beginnt.

Collatz-Problem



Algorithmenentwurf

1. Problem eingrenzen und (grob) formulieren

2. Problem analysieren und Modell(e) bilden:

- Problemspezifikation (Formalisierung)
- Wie könnte eine Lösung aussehen? Festlegung Datenstruktur(en)
- Gibt es überhaupt eine Lösung? Ist diese eindeutig?

3. Algorithmen entwerfen

- Lösungsverfahren aus Spezifikation ableiten
- gibt es eine Lösung gibt, gibt es unendlich viele
- erwünschte Eigenschaften: elegant, kurz, verständlich, transparent, schnell, leicht veränderbar

4. Verifikation, Test

5. Aufwandsanalyse, Komplexität abschätzen

6. Programm(e) entwickeln, testen und einsetzen

Beispiel: Algorithmenentwurf

1. Problemformulierung und -eingrenzung:

"Finde die jüngste Person im Raum" (Alle Personen? Was heißt *"jüngste"*?)

2. Problemanalyse: Abstraktion und Modellbildung:

"Alter"?
"jüngste"?
"finde"?
"Person"?

Beispiel: Algorithmenentwurf

1. Problemformulierung und -eingrenzung:

"Finde die jüngste Person im Raum" (Alle Personen? Was heißt *"jüngste"*?)

2. Problemanalyse: Abstraktion und Modellbildung:

"Alter"?	-> positive ganze Zahl (Datumseingabe)
"jüngste"?	-> Ordnungsrelation auf Zahlen
"finde"?	-> Altersangaben erfassen
"Person"?	-> Altersangaben a_1, ... , a_n mit Index (Person 1 ,..., n)

Beispiel: Algorithmenentwurf

1. Problemformulierung und -eingrenzung:

"Finde die jüngste Person im Raum" (Alle Personen? Was heißt *"jüngste"*?)

2. Problemanalyse: Abstraktion und Modellbildung:

"Alter"?	-> positive ganze Zahl (Datumseingabe)
"jüngste"?	-> Ordnungsrelation auf Zahlen
"finde"?	-> Altersangaben erfassen
"Person"?	-> Altersangaben $a_1, \ldots, a_n$ mit Index (Person 1 ,..., n)

Problemspezifikation:

Gegeben:	Folge $a_1, \ldots, a_n$ von positiven Ganzzahlen
Gesucht:	Positionsindex j mit $a_j = \min \{a_1, \ldots, a_n\}$

Beispiel: Algorithmenentwurf

1. Problemformulierung und -eingrenzung:

"Finde die jüngste Person im Raum" (Alle Personen? Was heißt *"jüngste"*?)

2. Problemanalyse: Abstraktion und Modellbildung:

"Alter"?	-> positive ganze Zahl (Datumseingabe)
"jüngste"?	-> Ordnungsrelation auf Zahlen
"finde"?	-> Altersangaben erfassen
"Person"?	-> Altersangaben $a_1, \ldots, a_n$ mit Index (Person 1 ,..., n)

Problemspezifikation:

Gegeben:	Folge $a_1, \ldots, a_n$ von positiven Ganzzahlen
Gesucht:	Positionsindex j mit $a_j = \min \{a_1, \ldots, a_n\}$

Ist das Problem lösbar?
Ist die Lösung eindeutig?

Möglichkeiten der Darstellung

- natürliche Sprache
- Pseudo-Code (formale Sprache)
- Programmablaufplan (PAP)
- Struktogramm
- Programmiersprache

Beispiel: Algorithmenentwurf

3. Algorithmenentwurf in Pseudocode:

Beispiel: Algorithmenentwurf

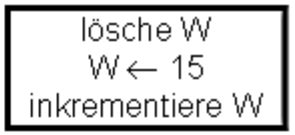
3. Algorithmenentwurf in Pseudocode:

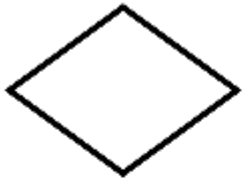
Verwende: n, i, j, min, $a_1..a_n$: Ganzzahl

(1) Lese n
(2) Setze j := 1 und min := a_j (* Dies ist das Ergebnis, falls n=1*)
(3) Falls n > 1, setze i := 2 (* Suchlauf *)
Solange i ≤ n wiederhole:
falls a_i < min, setze j := i und min := a_j
erhöhe i um 1
(4) Liefere j als Ergebnis

Ausgewählte Symbole - Programmablaufplan (PAP) nach DIN66001

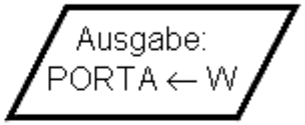
Operation, Anweisung allgemein





Verzweigung

Eingabe, Ausgabe



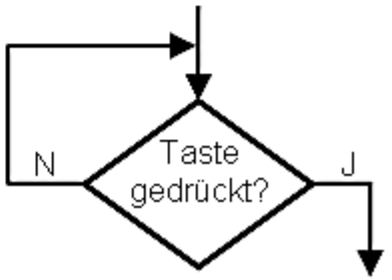
Unterprogramm

Grenzstelle (z. B.: START, STOP, RETURN, usw.)

Übergangsstelle, zur Kennzeichnung von Verbindungen zwischen Programmteilen, die an verschiedenen Stellen dargestellt sind.

Ablauflinien (Die Pfeilspitze kann zur Verdeutlichung der Ablaufrichtung verwendet werden)

Zusammenführung



Sprachen

Software - die Programmausstattung

- Bearbeitungsvorschriften für Hardware
- Unterteilung Betriebssystem - Anwendersoftware

Programmieren

- Umsetzung **Algorithmus** in Computerprogramm
- Verwendung einer Programmiersprache

Programmiersprache

- Ausdrucksmittel Menschen -> Maschine
- klare Definition durch **Syntax** und **Semantik**
- Programmiersprache erlernen vs. Programmieren erlernen

Maschinensprache

- **Maschinensprache** = Satz von Befehlen für eine ganz bestimmte Rechenmaschine

- Programmieren in Maschinensprache ist mühsam und fehleranfällig

```
LOAD    b,R0
LOAD    c,R1
ADD     R0,R1
SUB     2,R1
STORE   R1,a
```

Höhere Programmiersprache

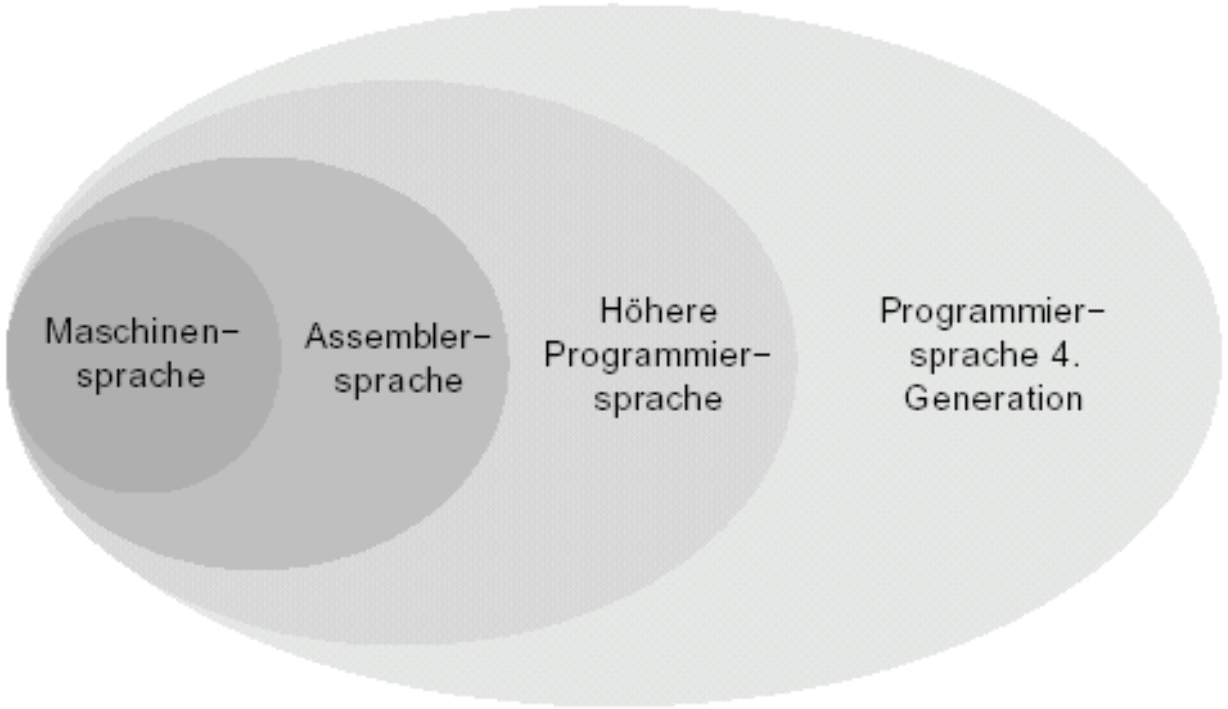
```
a = b + c - 2;
```

- Orientierung an **Problemen statt Maschinen**

- Vorteile:
  - schnellere und sichere Programmerstellung
  - bessere Lesbarkeit
  - Wiederverwendbarkeit (Portierbarkeit)

- Nachteil: Programme können nicht unmittelbar ausgeführt werden
  → Übersetzung notwendig (**Interpreter** und **Compiler**)

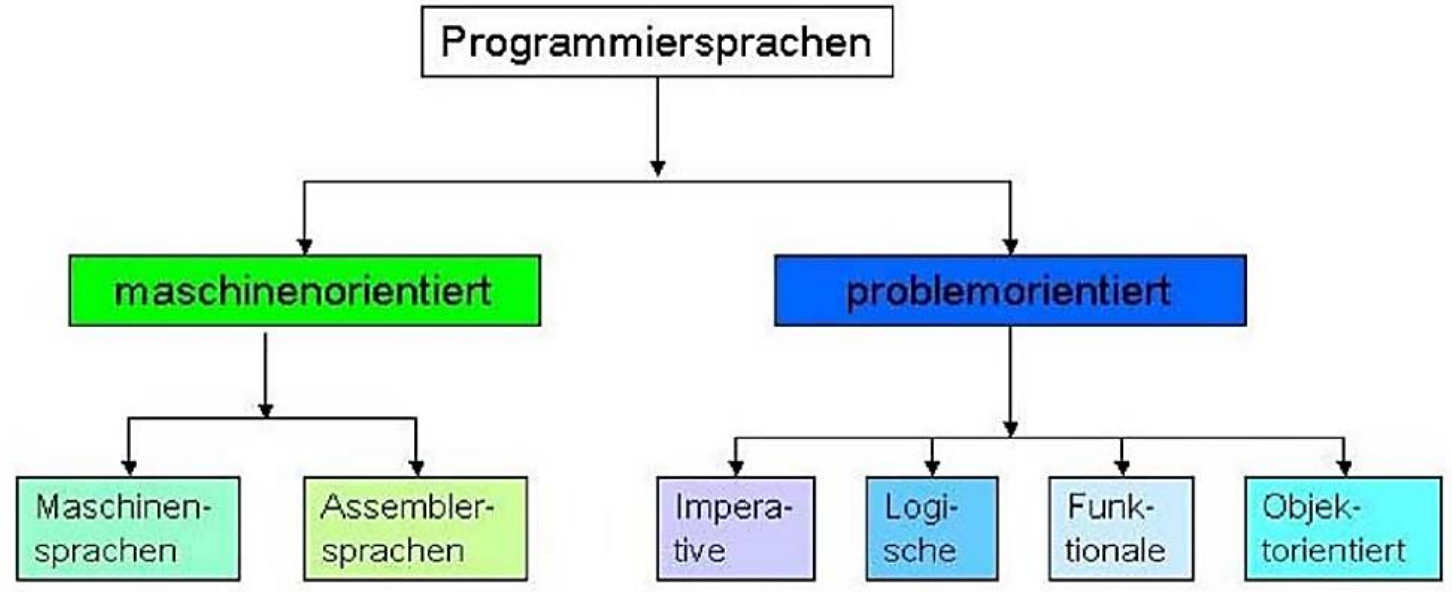
Sprachevolution



maschinenorientiert vs. problemorientiert

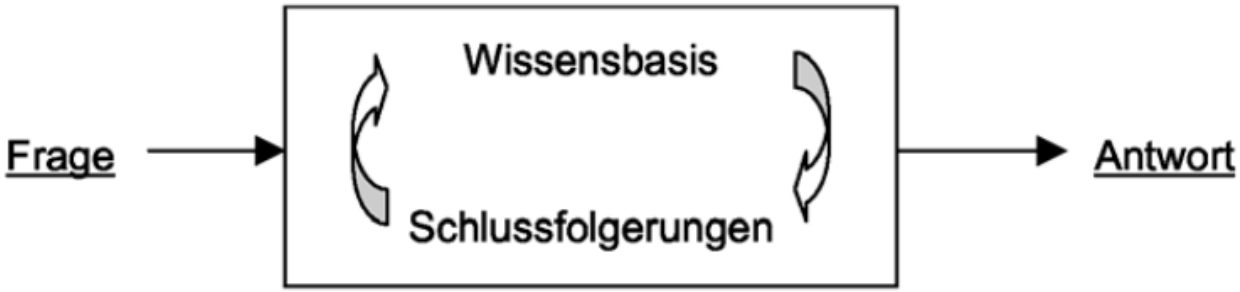
Definition nach DIN 44300:

> *"Eine Programmiersprache heißt problemorientiert, wenn sie geeignet ist, Algorithmen aus einem bestimmten Anwendungsbereich unabhängig von einer bestimmten Rechenanlage abzufassen und wenn sie sich an eine in dem betreffenden Bereich übliche Schreib- und Sprechweise anlehnt."*



Logische Sprachen

- Menge von Axiomen
- Spezifikation von Wissen und Annahmen
- keinerlei Steuerungsanweisungen
- "wissensbasierte Programmierung"



Imperative Programmiersprachen

- Sprachen der ersten bis dritten Generation

- Beschreibung von Programmabläufen (Funktionsorientierte Methode)

1) Problemlösung = Folge von Einzelschritten (-> **Algorithmus**)

```
BeginneProgramm

        Anweisung 1
        Anweisung 2
        Anweisung 3
        ...
BeendeProgramm
```

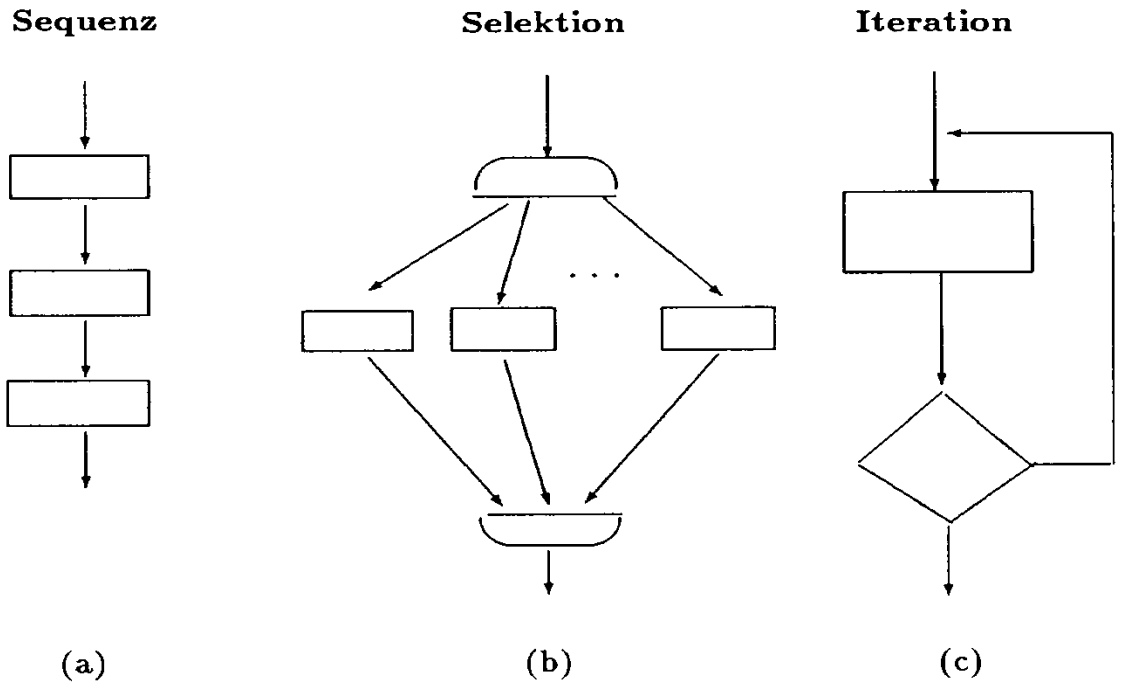
2) Programm = Vereinbarungen und Verarbeitungen (-> **Algorithmus**)

3) Anwendung des **Prinzips der Strukturierten Programmierung**

-> sprachenübergreifendes Programmierparadigma
-> Methoden und Regeln zum Entwurf von Algorithmen/Programmen
-> vollständige Aufteilung des Problems in Teilprobleme
-> Herabsetzung der Fehleranfälligkeit

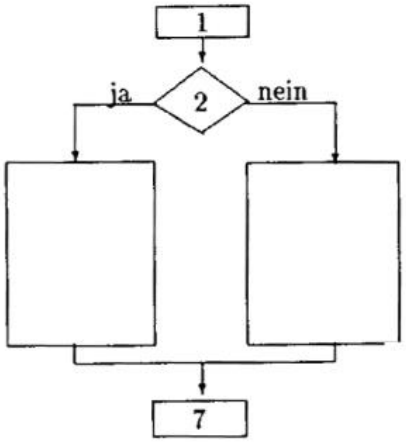
Prinzipien der Strukturierten Programmierung (1)

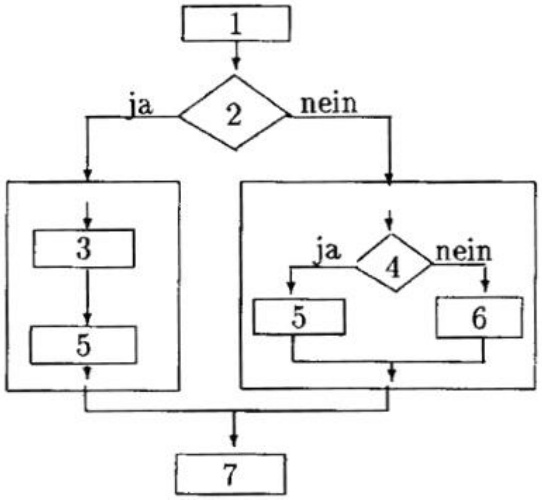
- Struktur-Theorem (Edsger W. Dijkstra 1972)
- hierarchische Programmorganisation (Trennung **Steuerung** - **Verarbeitung**)
- Beschränkung der Ablauf-Steuerung auf 3 Grundelemente:

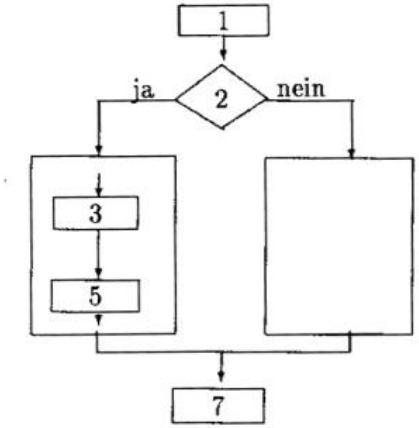


Prinzipien der Strukturierten Programmierung (2)

- Vorgehensweise: Top-Down Zerlegung, schrittweise Verfeinerung

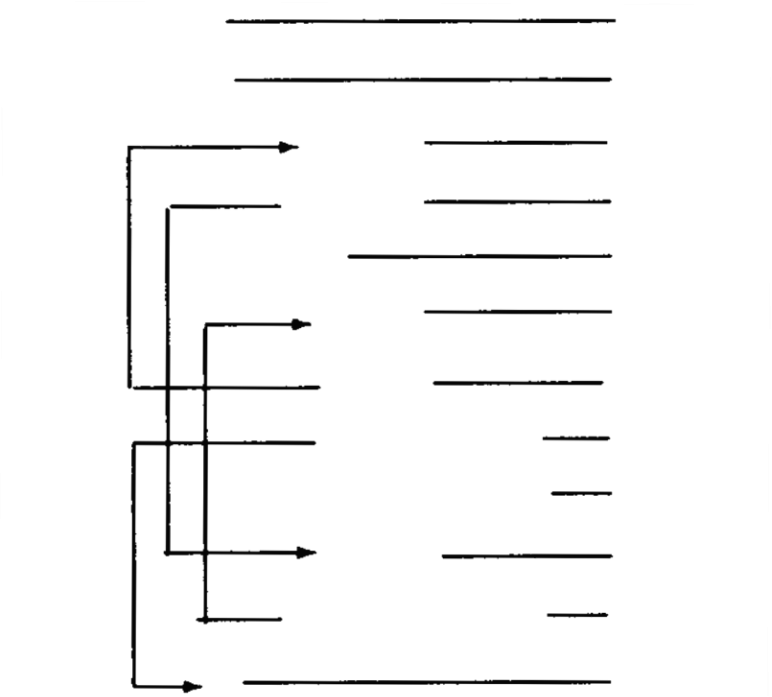






Prinzipien der Strukturierten Programmierung (3)

- Verzicht auf Sprünge ("GOTO")



abstrahiertes Programmbeispiel

Prozedurale Programmiersprachen

Prozedur/Unterprogramm: an anderer Stelle festgelegte Menge von Operationen

- Informationsaustausch mittels Parameterübergabe Ergebnisrückgabe
- Strukturierung bei wiederholter Verwendung von Programmteilen (mit gleichen oder veränderlichen Parametern)