Herzlich willkommen!

Dozent: Dipl.-Ing. Jürgen Wemheuer

Mail: wemheuer@ewla.de
Online: http://cpp.ewla.de/

Disclaimer

- Diese Vorlesungs-/Unterrichtsfolien wurden durch den Dozenten ausschließlich für die Gestaltung seines Unterrichts / seiner Vorlesung zusammengestellt bzw. verfasst und sind nicht als Referenz einer Programmiersprache gedacht.
- Das Skript verwendet teilweise Materialien meiner geschätzten Fachkollegen Prof. Dr. Dietrich Kuhn († 2010) und Prof. Dr. Stefan Enderle der Naturwissen-schaftlich-Technischen Akademie Dr. Grübler (nta) in Isny/Allgäu. Vielen Dank!
- Dem Vorlesungsskript mangelt es an jeglichem Kontext.
  Dieser ist vielmehr der bestimmende Lehrinhalt in den Vorlesungen.
- Nicht alle Inhalte des Vorlesungsskripts sind prüfungsrelevant.
- Nicht alle prüfungsrelevanten Fakten sind im Vorlesungsskript enthalten.
- Ausschlaggebend für Prüfungen sind deshalb allein die im Unterricht bzw. in den Übungen und/oder Projektbeispielen vorgebrachten Inhalte.
- Aktuelle Änderungen des Vorlesungsskripts sind jederzeit vorbehalten.
- Mit allen auftretenden Fragen zum Fachgebiet und dem Vorlesungsskript sollten sich die SchülerInnen und StudentInnen stets an den Dozenten wenden.
- Das Vorlesungsskript wurde mit bestem Wissen und Gewissen und sorgfältig erarbeitet, jedoch können Irrtümer und Fehler nicht ausgeschlossen werden.
- Jegliche Haftung und Gewährleistung ist ausgeschlossen.

Programmiertechnik

Teil 1:

- Literaturhinweise
- Einführung
- Grundlagen:
  - Prozessor – Speicher – Bus
  - ASCII-Codierung
  - Maschinensprache und Assembler
  - Compiler-Sprachen
  - Prozedur – Objekt
  - Editieren – Kompilieren – Linken
  - Die 4 Phasen der Programmerstellung
  - Programmentwicklung mit C/C++

Literaturempfehlungen

- Herbert Schildt:
  C++ IT-Tutorial, mitp-Verlag Bonn, 2003, ISBN 3-8266-0980-8
- Thomas Strasser: C++ - Programmieren mit Stil. Eine systematische Einführung, dpunkt-Verlag, 2003, ISBN 3-89864-221-6
- Rainer Krienke: C++ kurzgefasst. Eine Einführung in C++, Spektrum Akademischer Verlag Heidelberg/Berlin, 1999, ISBN 3-8274-0374-X
- Erlenkötter, Helmut:
  C++ Objektorientiertes Programmieren von Anfang an, 12. Auflage 2008, Rowohlt, Reinbek, ISBN 978-3-499-60077-7 (10,95 €)
- Liberty, Jesse: C++ in 21 Tagen, Markt+Technik, München (vergriffen)
- Louis, Dirk: C++ - Programmieren mit einfachen Beispielen, 2009, Markt+Technik, München, ISBN 978-3-8272-4483-3
  (352 Seiten - 1 CD, 2-farbig, 12,95 €)

Übung: Recherchieren Sie im Internet nach (kostenlosen) Online-Tutorials / E-Books

Einführung: Vorkenntnisse?

- Welche Erfahrungen haben Sie mit
  - Computern
  - anderen programmierbaren Geräten?
- Haben Sie schon einmal programmiert?
- Kennen („sprechen“) Sie eine/mehrere Programmiersprachen?
  - Wenn ja: welche?

Einführung: Programmieren

- Was ist „*Programmieren*“?
  - Erstellung von Computerprogrammen...
  - Was ist ein Programm?
- **Programm:**
  - *Anweisungen für einen Computer*
- **Programmieren:**
  - *Einem Computer sagen, was er tun soll...*

Mittelwert von Einkäufen bilden

1. Nimm drei Zettel, schreibe auf den ersten „Summe“, auf den zweiten „Anzahl“ und auf den dritten „Maximum“
2. Schreibe unter den Titel auf jeden der drei Zettel eine Null
3. Wenn ein Kunde eingekauft hat, merke dir seinen Rechnungsbetrag
4. Addiere den Rechnungsbetrag zur Zahl auf dem ersten Zettel und ersetze diese Zahl durch die Summe, die du berechnet hast
5. Ersetze die Zahl auf dem zweiten Zettel durch die um eins größere Zahl
6. Wenn der Rechnungsbetrag größer ist als die Zahl auf dem dritten Zettel, ersetze die Zahl auf dem dritten Zettel durch den Rechnungsbetrag
7. Solange noch ein Kunde kommt, mache weiter bei 3.
8. Dividiere die Zahl auf dem ersten Zettel durch die Zahl auf dem zweiten Zettel und gib die so berechnete Zahl als Mittelwert aus.
9. Gib die Zahl auf dem dritten Zettel als maximale Kaufsumme aus
10. Mache Feierabend

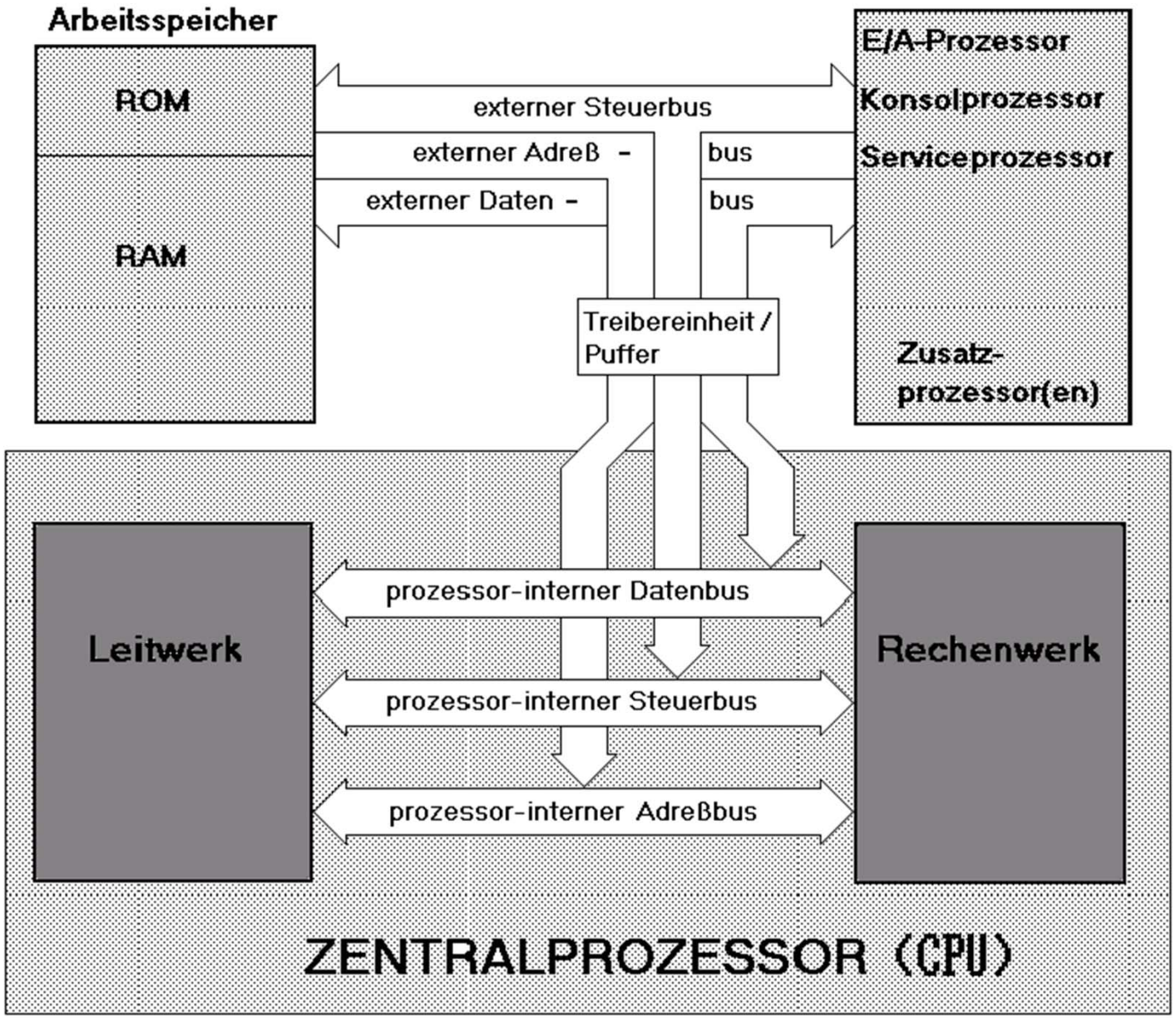
Einführung: Programmieren

- Wie sag‘ ich‘s meinem Computer?
  - Damit er mich versteht...

Dazu müssen wir den Computer etwas besser kennen lernen:

- *Hardware („Wie funktioniert er?“)*
- *Software („Wie spricht er?“)*

Grundstruktur des „frei programmierbaren elektronischen Ziffernrechners“

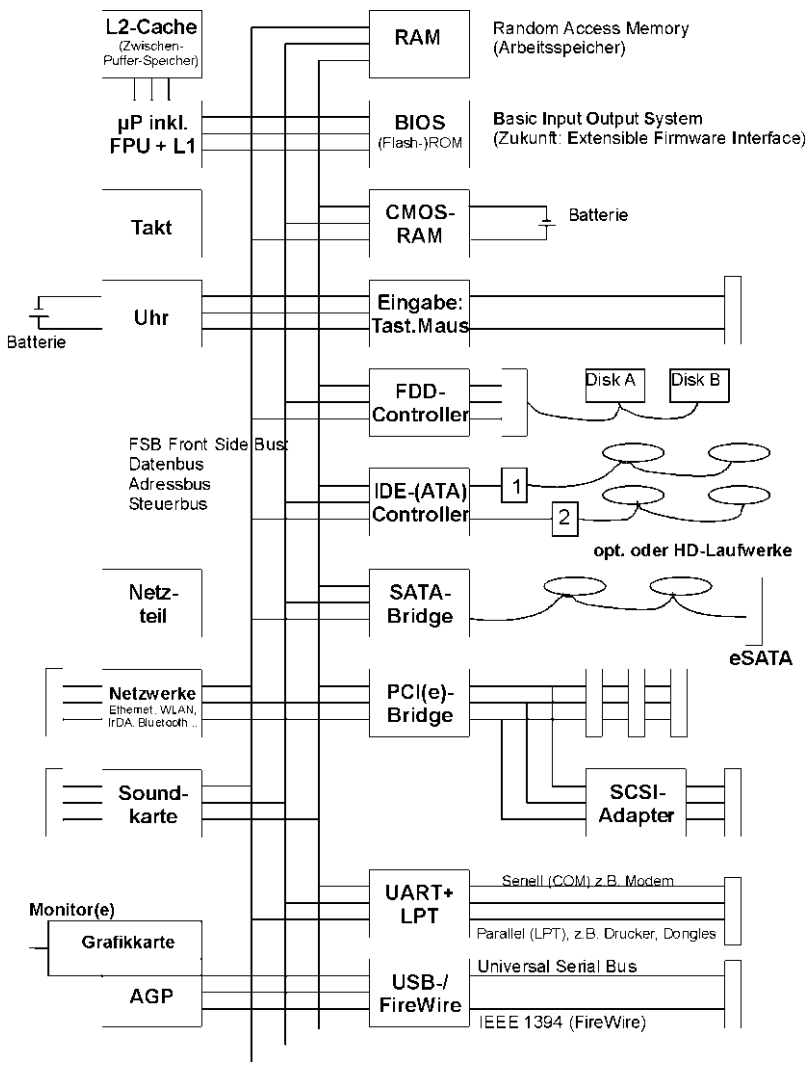


Der PC intern: Prinzip-Struktur

Alle Bauteile sind durch den

- Datenbus
- Adressbus
- Steuerbus

miteinander verbunden und tauschen hierüber Daten aus.



Das Byte: dual, oktal, hexa-/dezimal

Zweierpotenz	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Dez. Wertigkeit	128	64	32	16	8	4	2	1
Dualzahl	1	0	1	0	1	0	1	0
Hexadezimal	A				A			
Oktal	2			5			2	
Dezimal	170							

Was sind Daten?

Daten sind:

Gebilde aus Zeichen oder kontinuierliche Funktionen, die

- zum Zweck der Verarbeitung
- aufgrund bekannter oder unterstellter Vereinbarungen

Informationen darstellen.

(DIN ISO/IEC 2382)

Woraus bestehen Daten?

Daten sind im Allgemeinen in der Informationstechnik gekennzeichnet durch:

- eine Bezeichnung
  (Bezeichner, Name)
- einen Datentyp
  (Ganzzahl, Dezimalzahl, Zeichen, Text etc.)
- einen Wert
  (die informationstechnisch codierte Information an sich)
  OBACHT: „Nichts“ ist nicht der Codewert „0“ sondern NULL

Zum Sprachgebrauch

lat.: dare (geben), datum = das Gegebene

Ein „Datum“ wird umgangssprachlich bevorzugt verwendet als Bezeichnung für einen „bestimmten Tag im Kalender“.

In der Informationstechnik ist „Datum“ als Singular für „Daten“ sehr geläufig, für den Kalendertag verwenden wir dann zur Unterscheidung besser den Begriff „Tagesdatum“ oder „Kalenderdatum“.

Datum: Bezeichnung

Bezeichner (Namen) werden benötigt, um in einem Rechnersystem ein bestimmtes Datum abzulegen und wiederzufinden (der Rechner wandelt diese Bezeichnung in einen Zeiger auf die entsprechende physikalische Speicherstelle um).

Bei einem einmalig auftretenden Datum reicht die Vergabe eines Namens (z.B. „MeinDatum“), bei ein- oder mehrdimensionalen Daten („Tupel“, „Matrix“) werden ein oder mehrere (i.d.R. durch Pfeil- oder Punktoperatoren abgetrennte) Namensbestandteile (z.B. Schueler->Vorname, Schueler.Nachname) und/oder ein Index oder mehrere Indizes hinzugefügt (z.B. Listeneintrag[20], SpielfeldMatrix[8][8]).

Datum: Datentyp und Werte

Datentypen bestimmen:

- welchen Speicherplatz (Größe, Anzahl Bytes) ein Datum physikalisch (im Arbeitsspeicher, auf einem Speichermedium) beansprucht
- wie ein Wert intern codiert wird:
  4 als Ganzzahl: 0000 0000 0000 0100 (2 Byte Short Integer)
  4 als ASCII-Zeichen: 0011 0100 (1 Byte Character)
  4 als Dezimalzahl: 0100 0000 1000 0000 0000 0000 0000 0010
- wie ein Wert zu interpretieren ist und verarbeitet werden kann: **Zahl 5** multipliziert mit **Zeichen Y** ist unsinnig…*(?)*
- den Wertebereich, den ein Datum annehmen kann:
  (z.B. unsigned short int für Ganzzahlen von 0…65535)

Codierung: ASCII

ASCII:

American Standard Code for
Information Interchange

=> genormter 7-bit-Code,
8. Bit = Prüfbit

Information: =>
ist die inhaltliche Bedeutung der
von einem Sender abgehenden
Daten für den Empfänger

ASCII-Tabelle (000-255)

ASCII-Tabelle (MS-DOS Codeseite 437 (Englisch)

0		1	☺	2	☻	3	♥	4	♦	5	♣	6	♠	7	•	8	◘	9		10		11	♂	12	♀
12	♀	13		14	♫	15	☼	16	►	17	◄	18	↕	19	‼	20	¶	21	§	22	▬	23	↨	24	↑
24	↑	25	↓	26	→	27	←	28	∟	29	↔	30	▲	31	▼	32		33	!	34	"	35	#	36	$
36	$	37	%	38	&	39	'	40	(| 41 |)	42	*	43	+	44	,	45	-	46	.	47	/	48	0
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7	56	8	57	9	58	:	59	;	60	<
60	<	61	=	62	>	63	?	64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G	72	H
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O	80	P	81	Q	82	R	83	S	84	T
84	T	85	U	86	V	87	W	88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_	96	`
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g	104	h	105	i	106	j	107	k	108	l
108	l	109	m	110	n	111	o	112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w	120	x
120	x	121	y	122	z	123	{	124	|	125	}	126	~	127	⌂	128	Ç	129	ü	130	é	131	â	132	ä
132	ä	133	à	134	å	135	ç	136	ê	137	ë	138	è	139	ï	140	î	141	ì	142	Ä	143	Å	144	É
144	É	145	æ	146	Æ	147	ô	148	ö	149	ò	150	û	151	ù	152	ÿ	153	Ö	154	Ü	155	¢	156	£
156	£	157	¥	158	₧	159	ƒ	160	á	161	í	162	ó	163	ú	164	ñ	165	Ñ	166	ª	167	º	168	¿
168	¿	169	⌐	170	¬	171	½	172	¼	173	¡	174	«	175	»	176	░	177	▒	178	▓	179	|	180	┤
180	┤	181	╡	182	╢	183	╖	184	╕	185	╣	186	║	187	╗	188	╝	189	╜	190	╛	191	┐	192	└
192	└	193	┴	194	┬	195	├	196	─	197	┼	198	╞	199	╟	200	╚	201	╔	202	╩	203	╦	204	╠
204	╠	205	═	206	╬	207	╧	208	╨	209	╤	210	╥	211	╙	212	╘	213	╒	214	╓	215	╫	216	╪
216	╪	217	┘	218	┌	219	█	220	▄	221	▌	222	▐	223	▀	224	α	225	ß	226	Γ	227	π	228	Σ
228	Σ	229	σ	230	µ	231	τ	232	Φ	233	Θ	234	Ω	235	δ	236	∞	237	ø	238	ε	239	∩	240	≡
240	≡	241	±	242	≥	243	≤	244	⌠	245	⌡	246	÷	247	≈	248	°	249	∙	250	·	251	√	252	ⁿ
252	ⁿ	253	²	254	■	255																			

Einschub: Medientheorie

Wissen	Wissen
↕ Pragmatische Codierung	↕
Information	Information
↕ Semantische Codierung	↕
Nachricht	Nachricht
↕ Syntaktische / technische Codierung	↕
Signal ↔ *Übertragung mit Störungen* ↔	Signal

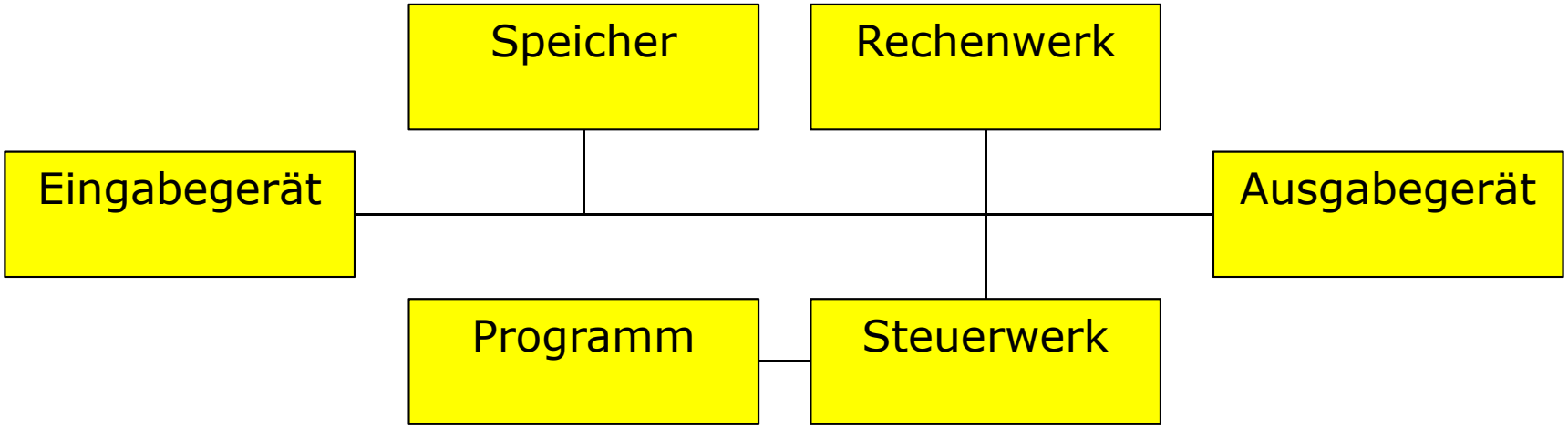
Codierungen aus der Medientechnik

Medium	Aufnahme Erzeugung	Speicherung Bearbeitung (Gestaltung) Verarbeitung	Distribution (online, offline) Wiedergabe
Text (+ textliche Repräsentation von Daten/Zahlen)	A: Scanner mit OCR (+ Barcode, RFID etc.) E: Tastatur und Maus	Office-Programme Textanalyse-Programme (Skript, Programmiersprachen)	Print Display / Projektion
Ton (Audio)	A: Mikrofon E: Synthesizer etc.	Audio-Editoren Sprachanalyse etc.	Hörfunk, Tonträger (analog und digital) Audiodateien (File / Stream) Lautsprecher, Kopfhörer
Standbild (Foto/Grafik)	A: Fotoapparat (analog/digital), Scanner E: Grafik (Pixel-/Vektorgrafik)	Bildbearbeitungsprogramme (Photoshop & Co) Bildanalyse etc.	Print, Belichter, Plotter Bilddateien Display / Projektion
Bewegtbild (Video/Animation)	A: Film- / Videokamera, Filmabtaster (Scanner) E: Computeranimation, Simulation/Labor	Videobearbeitungsprogramme, Videoschnitt Videoanalyse etc.	Fernsehen, VHS/DVD etc. (analog und digital) Videodateien (File / Stream) Display / Projektion
Multimedia	(siehe einzelne „Assets”) (+ Dramaturgie, Gestaltung)	Autorensysteme/Webeditoren, Mediensteuerung, -Ablauf „Hardcore”-Programmierung	s.o., zusätzliche „Lernlogik” (integriert, interaktiv)
Codierung	A-D-Umsetzung	Datenkompression, -Expansion Datenreduktion	D-A-Umsetzung

Was ist Software?

Wikipedia:

„Sammelbegriff für die Gesamtheit ausführbarer Programme und die zugehörigen Daten. Sie dient dazu, Aufgaben zu erledigen, indem sie von einem Prozessor ausgewertet wird und so softwaregesteuerte Geräte, die einen Teil der Hardware bilden, in ihrer Arbeit beeinflusst.“



Das EVA-Prinzip

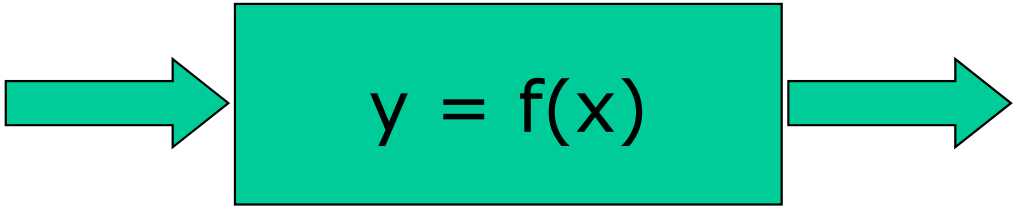
hier rein ->

-> da raus

E (Input) -> V (Processing) -> A (Output)

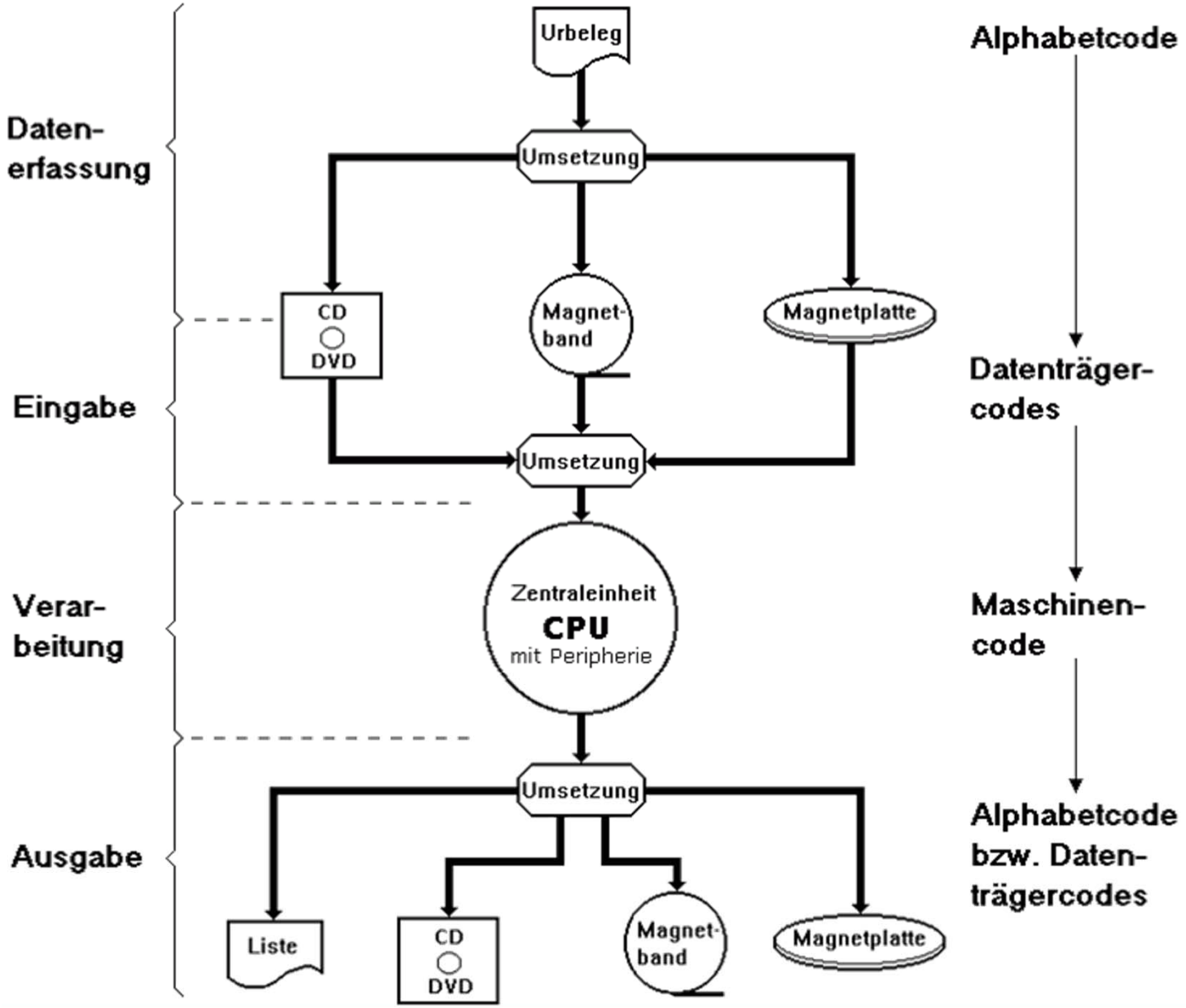
Die „Black Box“

- Das EVA-Prinzip gilt eigentlich überall in der Technik...
- ... und in der Mathematik...

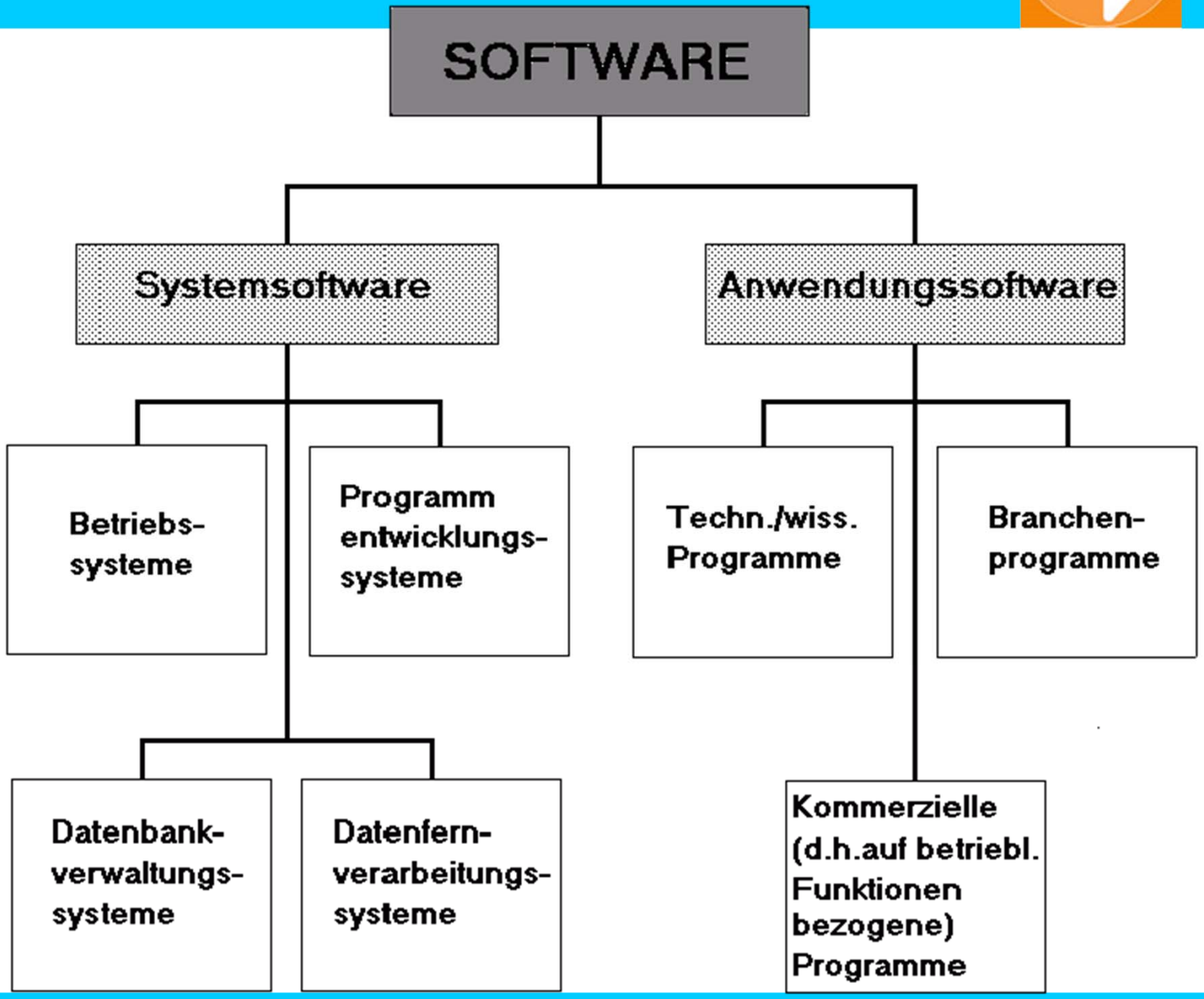


- ... und in Biologie, Natur, Gesellschaft, Politik, Psychologie, etc...?

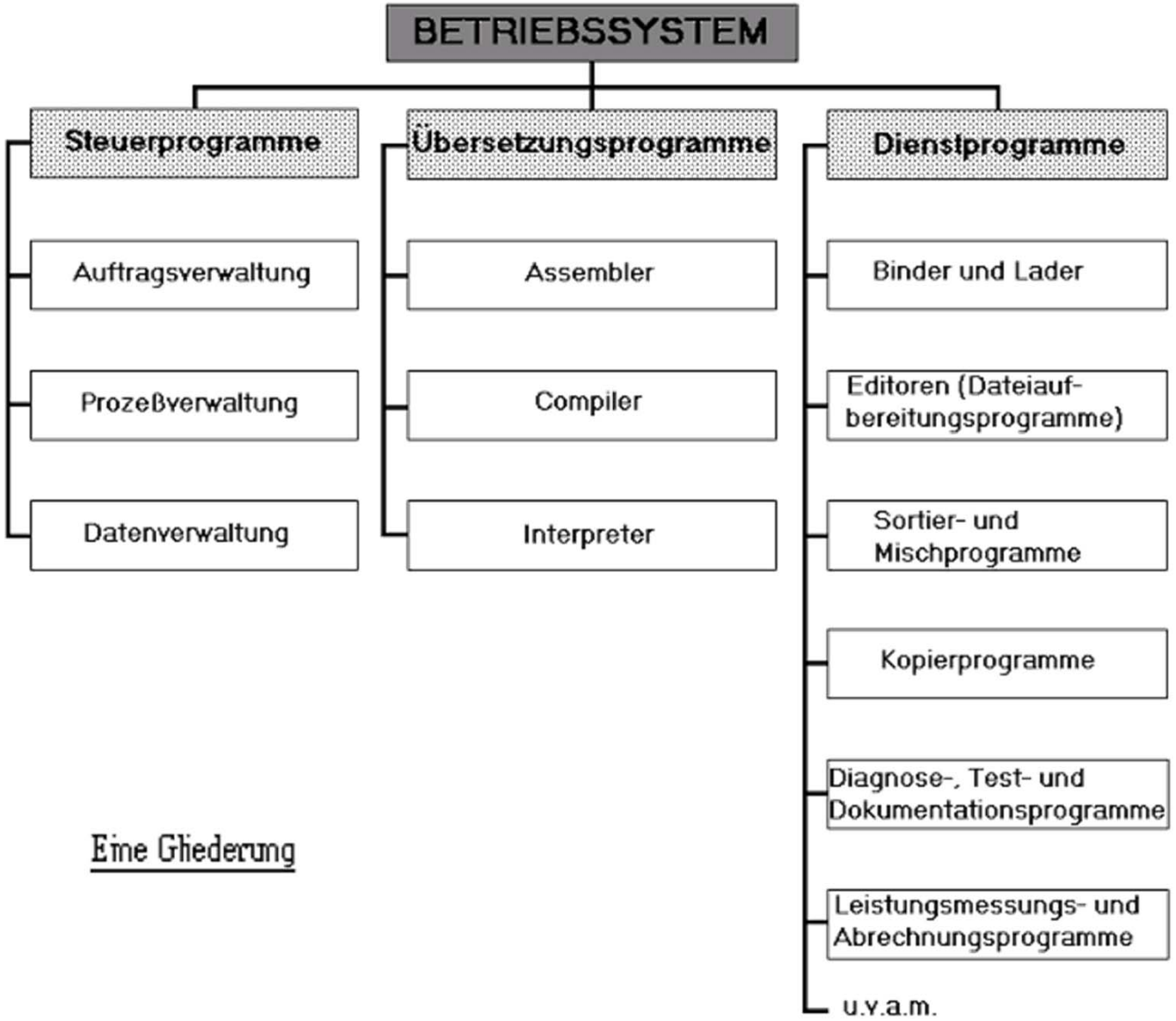
Datenfluss durch den Computer



Software: Einteilung

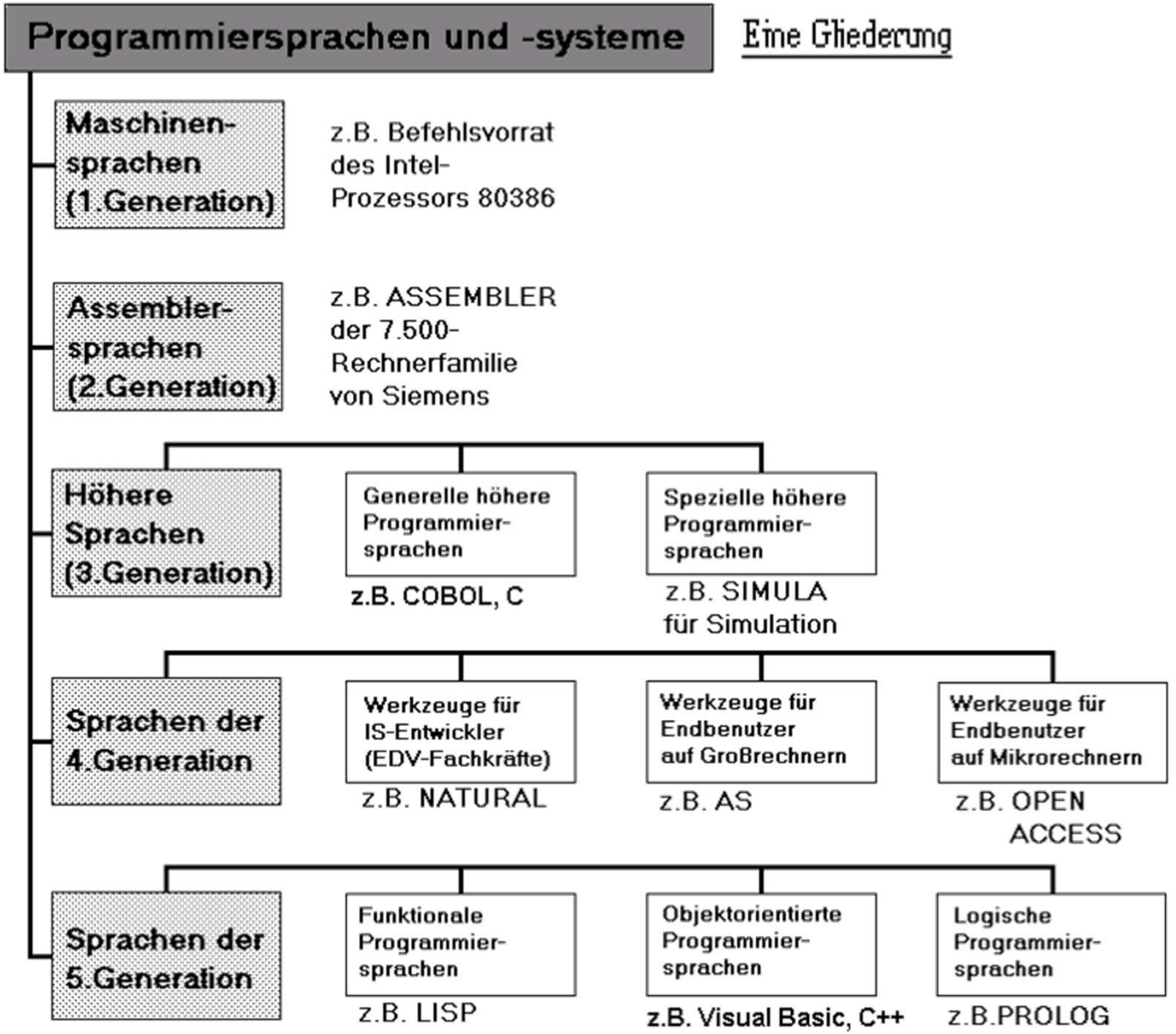


Aufgaben der Betriebssysteme



Eine Gliederung

Programmiersprachen: Einteilung



Befehle sind auch „Daten“...

- Einige Binärdaten werden als Anweisungen (Befehle) vom Prozessor ausgeführt:
  - ➢ Operatoren
- Einige Binärdaten werden als codierte Inhalte (Daten, Informationen) behandelt:
  - ➢ Operanden
- Beispiel:
  Beim Ausdruck **1 + 2** sind **1** und **2** die Operanden, die über den Operator **+** verknüpft sind

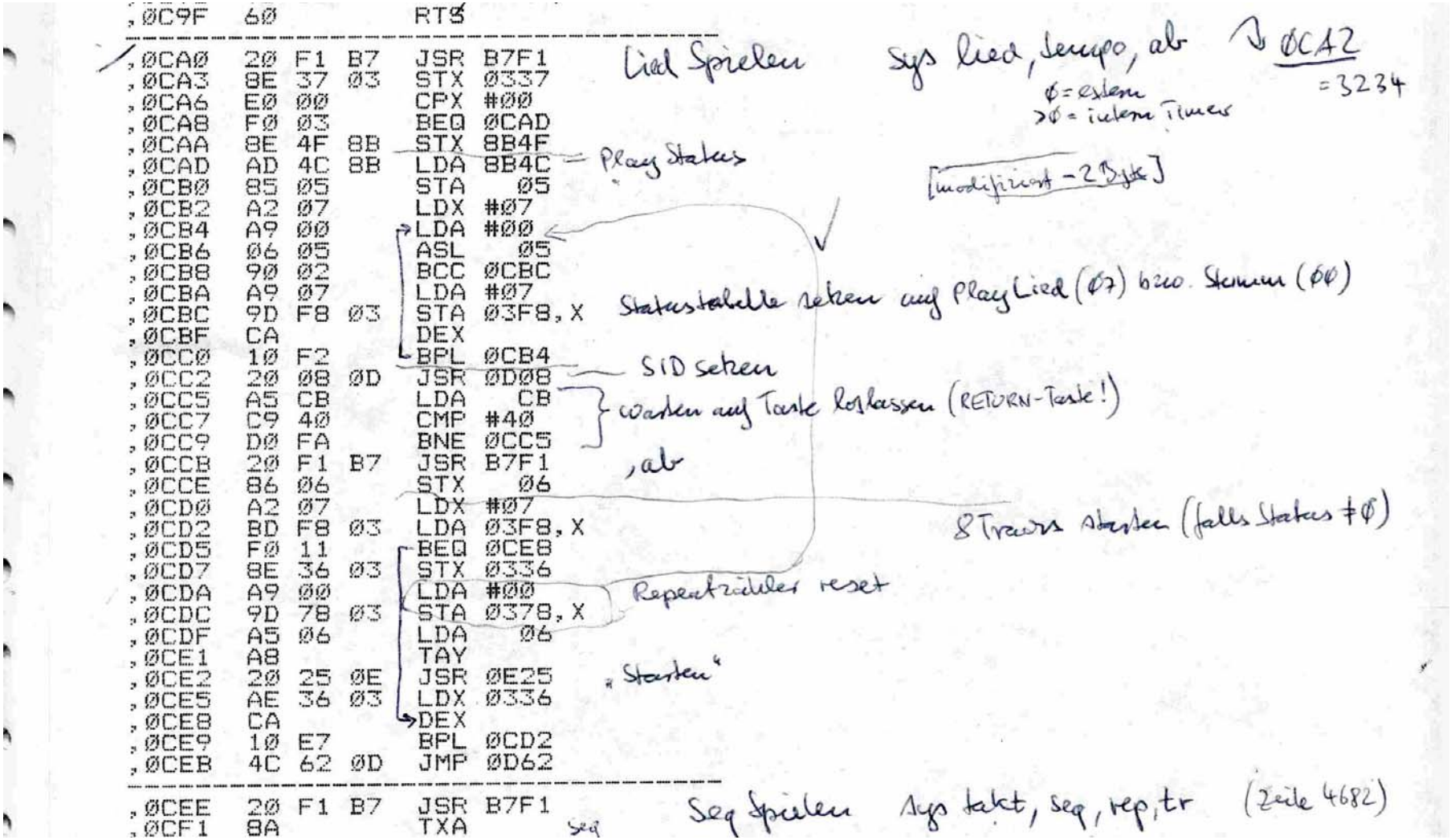
Assembler / Maschinensprache

Beim 6502-Prozessor (1980) war der Maschinencode noch übersichtlich (Auszug):

ADC	Add Memory to Accumulator with Carry	Addiere Werte aus Speicher und Rechenregister
AND	"AND" Memory with Accumulator	Logische UND-Verknüpfung von Rechenregister und Speicher
ASL	Shift Left One Bit (Memory or Accumulator)	Verschiebung eines Speicherinhaltes um eine Stelle nach links
BCC	Branch on Carry Flag Ciear	Verzweige, wenn im Statusregister kein Carry-Flag gesetzt ist
BCS	Branch on Carry Flag Set	Verzweige, wenn im Statusregister das Carry-Flag gesetzt ist
BEQ	Branch on Result Zero	Verzweige, wenn Ergebnis der vorherigen Operation Null ist
BMI	Branch on Result Minus	Verzweige, wenn Ergebnis der vorherigen Operation negativ ist
BNE	Branch on Result not Zero	Verzweige, wenn Ergebnis der vorherigen Operation nicht 0 ist
BPL	Branch on Result Plus	Verzweige, wenn Ergebnis der vorherigen Operation positiv ist
BRK	Force Break	Halte das Programm an!
BVC	Branch on Overflow Flag Clear	Verzweige, wenn im Statusregister kein Übertrag erfolgt ist
BVS	Branch on Overflow Flag Set	Verzweige, wenn im Statusregister ein Übertrag erfolgt ist
CLC	Clear Carry Flag	Setzte das Carry-Flag zurück auf Null
CLD	Clear Decimal Mode	Setze den Dezimal-Rechenmodus zurück
CLI	Clear Interrupt Disable Bit	Erlaube Interrupts
CLV	Clear Overflow Flag	Setze das Übertrags-Flag zurück auf Null
CMP	Compare Memory and Accumulator	Vergleiche Speicherinhalt mit Rechenregister
CPX	Compare Memory and Index X	Vergleiche Speicherinhalt mit Register X
CPY	Compare Memory and Index Y	Vergleiche Speicherinhalt mit Register Y
DEC	Decrement Memory by One	Subtrahiere vom Speicher den Wert Eins
DEX	Decrement Index X by One	Subtrahiere vom Register X den Wert Eins
DEV	Decrement Index Y by One	Subtrahiere vom Register Y den Wert Eins
EOR	"Exclusive-Or" Memory with Accumulator	Exklusiv-ODER-Verknüpfung von Rechenregister und Speicher
INC	Increment Memory by One	Addiere den Wert Eins zum Speicherinhalt
INX	Increment Index X by One	Addiere den Wert Eins zum Register X
INY	Increment Index Y by One	Addiere den Wert Eins zum Register Y
JMP	Jump to New Location	Spring an neue Adresse
JSR	Jump to Subroutine Saving Return Address	Spring an neue Adresse, merke Rücksprungadresse
NOP	No Operation	Tue nichts
...		...

Assembler-Beispiel (Sequenzer)

Wer blickt da noch durch???



Assembler: Vor- und Nachteile

- Argumente für Assembler-Programmierung:
  - Geschwindigkeits-Optimierung (nur bedingt)
  - Speicherplatz-Optimierung
  - Verwendung von Spezialbefehlen (MMX, SSE)
  - Mangel an Compilern (für spezielle Prozessoren)
- Argumente gegen Assembler-Programmierung:
  - Entwicklungs- und Wartungsaufwand
  - Maschinenabhängigkeit
  - Betriebssystem-Abhängigkeit
  - Assembler-Abhängigkeit (Syntax)

Befehlstypen

- Transportbefehle:
  - Holen, Laden: => Kopieren!
  - Schreiben: => Überschreiben!
  - Löschen: gibt es nicht...
- Rechenbefehle:
  - Arithmetische und logische Ausdrücke
- Kontrollstrukturen:
  - Funktionsaufrufe
  - Bedingungen (if / switch)
  - Schleifen (for, while)

Programmiersprache C

- C:
  - entwickelt ca. 1971 von Dennis Ritchie, Bell Labs
    (um das UNIX-Betriebssystem schreiben zu können)
- K&R C:
  - 1978 von D. Ritchie und Brian W. Kernighan
    (damit man mit C auch mal Anwendungen programmieren kann)
- ANSI C:
  - 1983-1989: Eine Norm muss her!
    (weil die Compiler sonst nicht wissen, was sie kompilieren sollen...)
- C99:
  - 1999 von der ISO, C++-Erkenntnisse fließen ein

Programmiersprache C++ (C#)

- C++:
  - entwickelt ca. 1980 von Bjarne Stroustrup, AT&T
  - Erweiterung von C um *Objektorientierung*
    - *=> C ist im Wesentlichen eine Untermenge von C++*
- C#:
  - Eigenentwicklung von Microsoft ® mit Sprachelementen aus C, Java, Delphi, C++, für die .NET-Strategie, zum Beispiel mit Berücksichtigung von PC-Benutzerrechten...

Einige Unterschiede C / C++

- Ein- und Ausgabe:
  - C: printf, fgets, ... C++: cin, cout, <<, >>
- Kommentare:
  - C: nur /* ... */ (ab C99 auch: // ...) C++: /* ... */ und // ...
- Bool:
  - C: Als Int (ab C99: Eigener Datentyp) C++: Eigener Typ
- Dyn. Speicherverwaltung:
  - C: malloc / free C++: new / delete
- Zeichenketten:
  - C: Feld von Einzelzeichen: char[] C++: Eigener Typ: string
- Headerdateien/Bibliotheken:
  - C: z.B. <math.h> C++: z.B. <cmath>

Programmiersprache C

Immer noch eine der am meisten eingesetzten Programmiersprachen
(UNIX, auch im Windows-BS und bei vielen „Apps“), wenn auch mit einigen Mankos behaftet...

- *Vorteile:*
  - *Effizient*
  - *Direkter Zugriff auf die Hardware*
  - *Schult das Computer-Verständnis*
- *Nachteile:*
  - *Wenig komfortabel*
  - *Viele Fehlermöglichkeiten (nicht "deppensicher")*
  - *Der Programmierer muss sich um Vieles selbst kümmern, z.B. Speicherplatzverwaltung...*

Typen von Programmiersprachen

- Imperative Sprachen
  - C, C++, ...
- Prozedurale Sprachen
  - C, Cobol, Fortran, Ada, ...
- Objektorientierte Sprachen
  - C++, Java, C#, Smalltalk, Visual Basic, viele Scriptsprachen
- Logische Sprachen
  - Prolog, ...
- Funktionale Sprachen
  - Lisp, ...

Programmierung

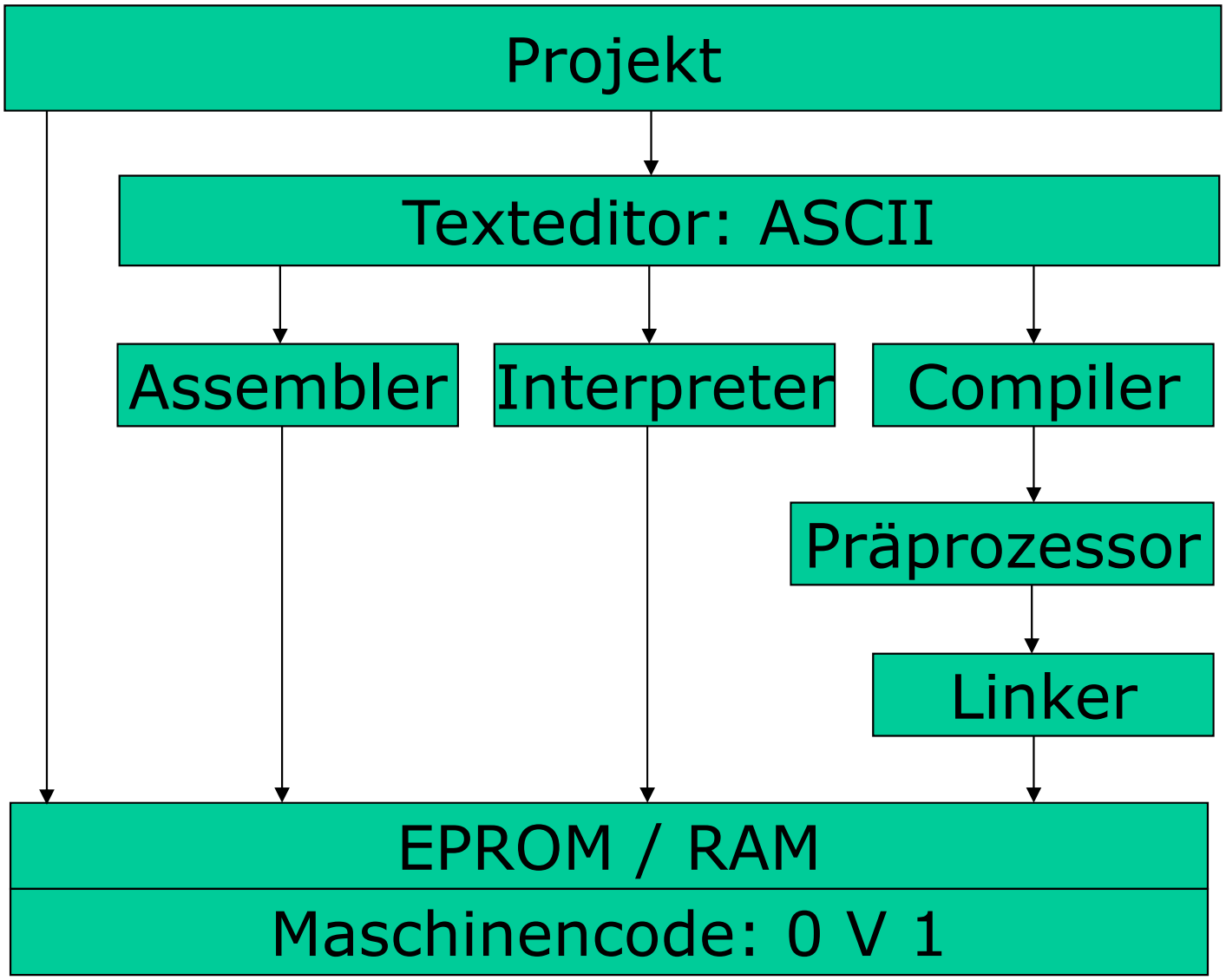
Prozedural

- ist "out" !!!
- die Programme sind eine Folge von auszuführenden Anweisungen auf einen Satz von Daten.
- mit der strukturierten Programmierung wird eine Systematik in die Organisation dieser Prozeduren und in die Verwaltung großer Datenmengen gebracht

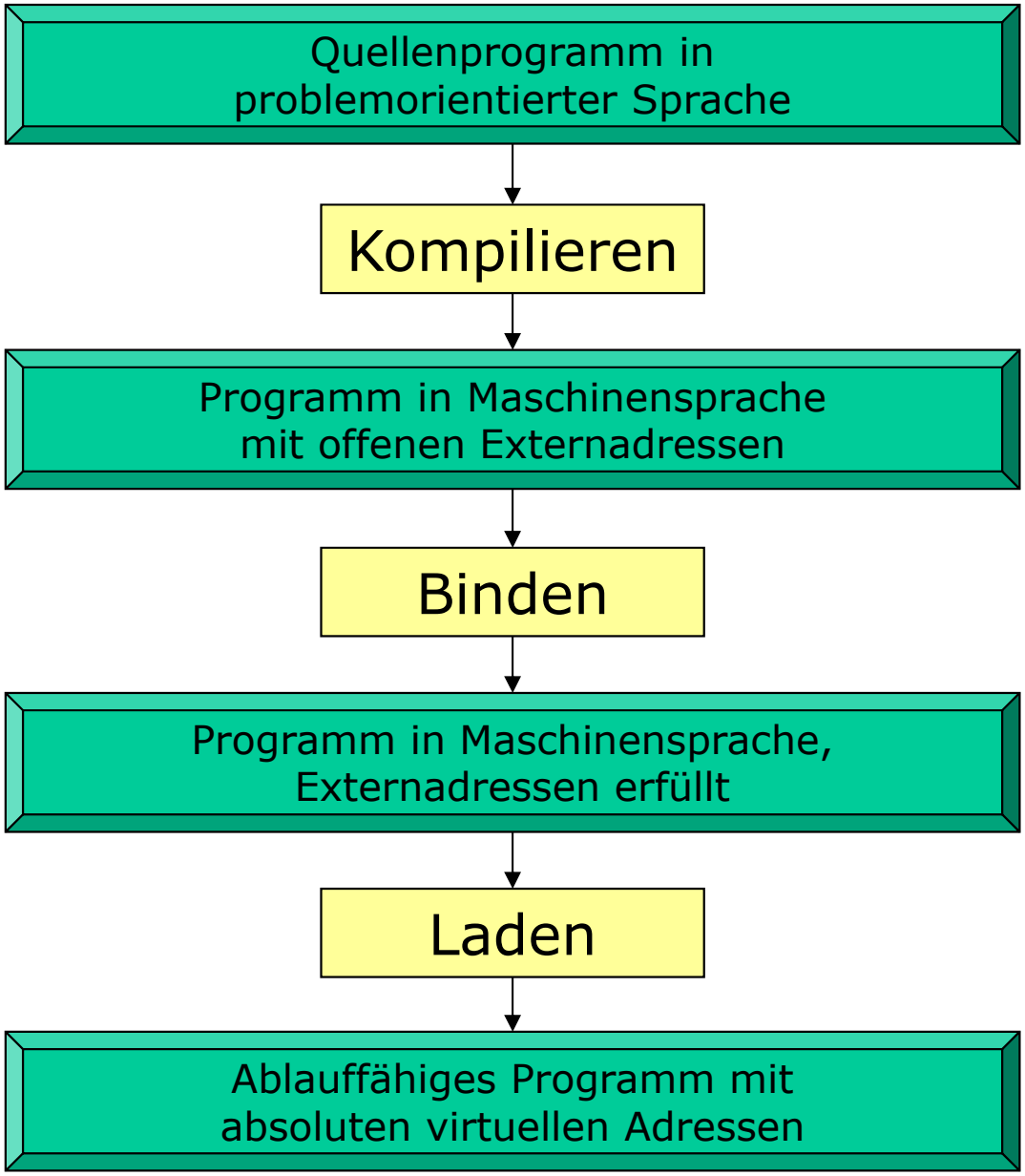
Objektorientiert

- ist "in" !!!
- die Daten (und die Prozeduren), die auf diesen Daten arbeiten, werden als geschlossenes OBJEKT behandelt.
- das Objekt ist eine selbstständige Einheit mit einer Identität und mit einem eigenen Charakteristikum.
- vier Säulen:
  1. Kapselung
  2. Verbergen von Daten
  3. Vererbung
  4. Polymorphie

Programmierwege



Programmierungstechnik

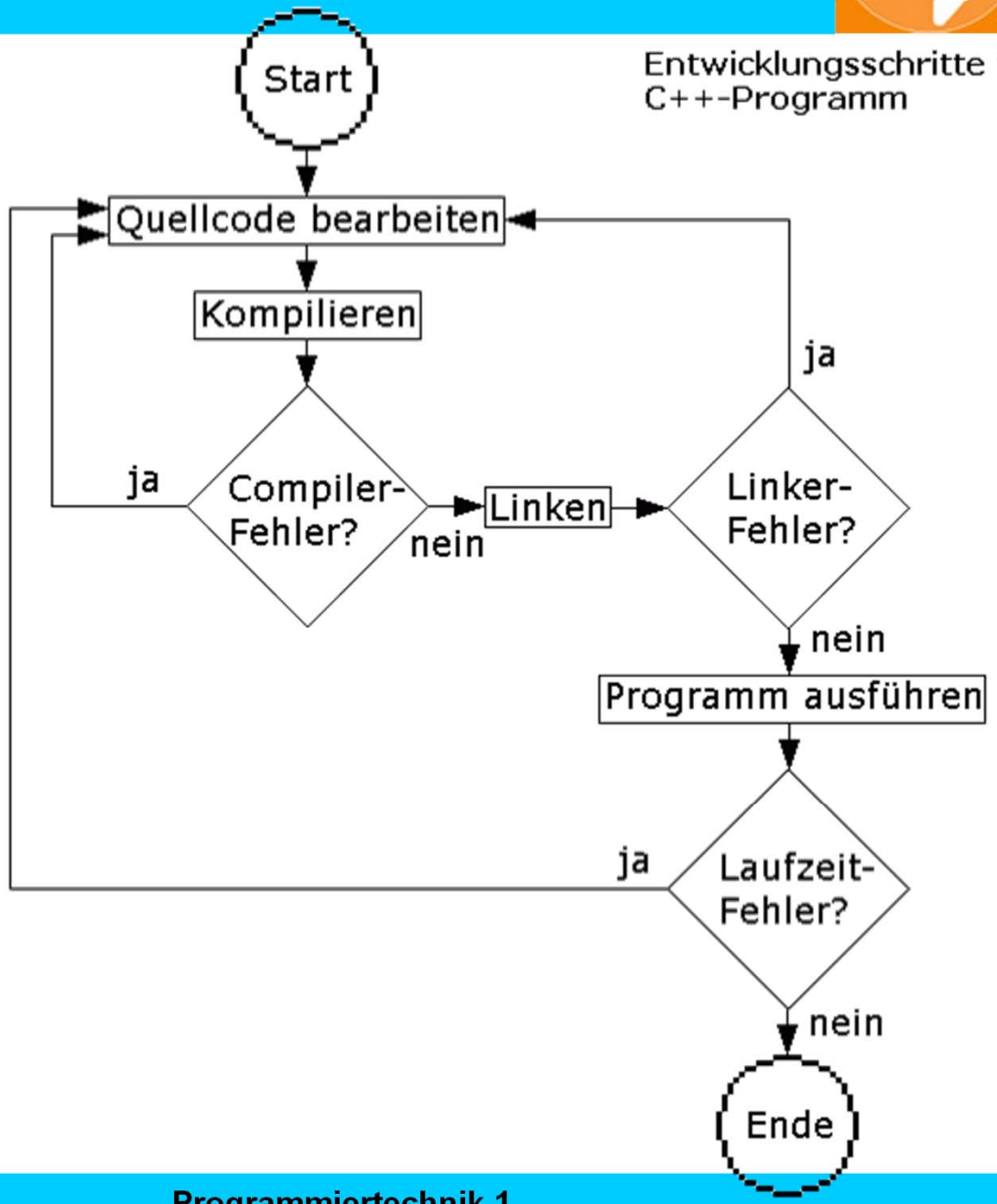


Technische Entwicklungsschritte

1. Schritt:
   Quellprogramm schreiben *(*.cpp)*
2. Schritt:
   Quellprogramm -> übersetzen (kompilieren)
   => Objektdatei
3. Schritt:
   Objektdatei + Bibliotheken -> binden (linken)
   => ausführbares Programm *(*.exe / *.dll)*

Frage: Was ist ein Cross-Compiler?

Technische Entwicklungsschritte



Phasen der Programmentwicklung

1. Problemanalyse
2. Programmentwurf
3. Erstellen des Quellprogramms
4. Testen des Programms

Problemanalyse

- Welches Problem ist zu lösen?
  - Beispiel Geschwindigkeitsberechnung
- Welche Angaben werden benötigt?
  - zurückgelegter Weg, benötigte Zeit
  - (Test, ob eingegebene Werte sinnvoll sind)
- Was soll das Programm tun?
  - Berechnung der Geschwindigkeit aus zurückgelegtem Weg und der dafür benötigten Zeit
  - v=s/t => t muss größer 0 sein!
- Ausgabe der berechneten Werte mit erläuterndem Text
  - z.B. "Die Geschwindigkeit beträgt ... km/h"

Programmentwurf

- Als nächstes wird die Aufgabe als **Algorithmus** formuliert.
  Der Algorithmus ist eine Verarbeitungsvorschrift,
  die angibt, wie Eingangsdaten schrittweise in Ausgangsdaten umgewandelt werden...
  ...unabhängig von der verwendeten Programmiersprache!
- Die Folge der Verarbeitungsschritte muss eindeutig festgelegt sein.
- Größere Probleme werden dabei in Teilaufgaben und Teilaspekte aufgeteilt.
  (Ob der Algorithmus tatsächlich auf dem Papier oder nur im Kopf des Programmierers entwickelt wird, hängt von der Komplexität der Aufgabe und der Genialität des Programmierers ab).

Algorithmus

- Der Algorithmus kann zunächst umgangssprachlich formuliert werden:

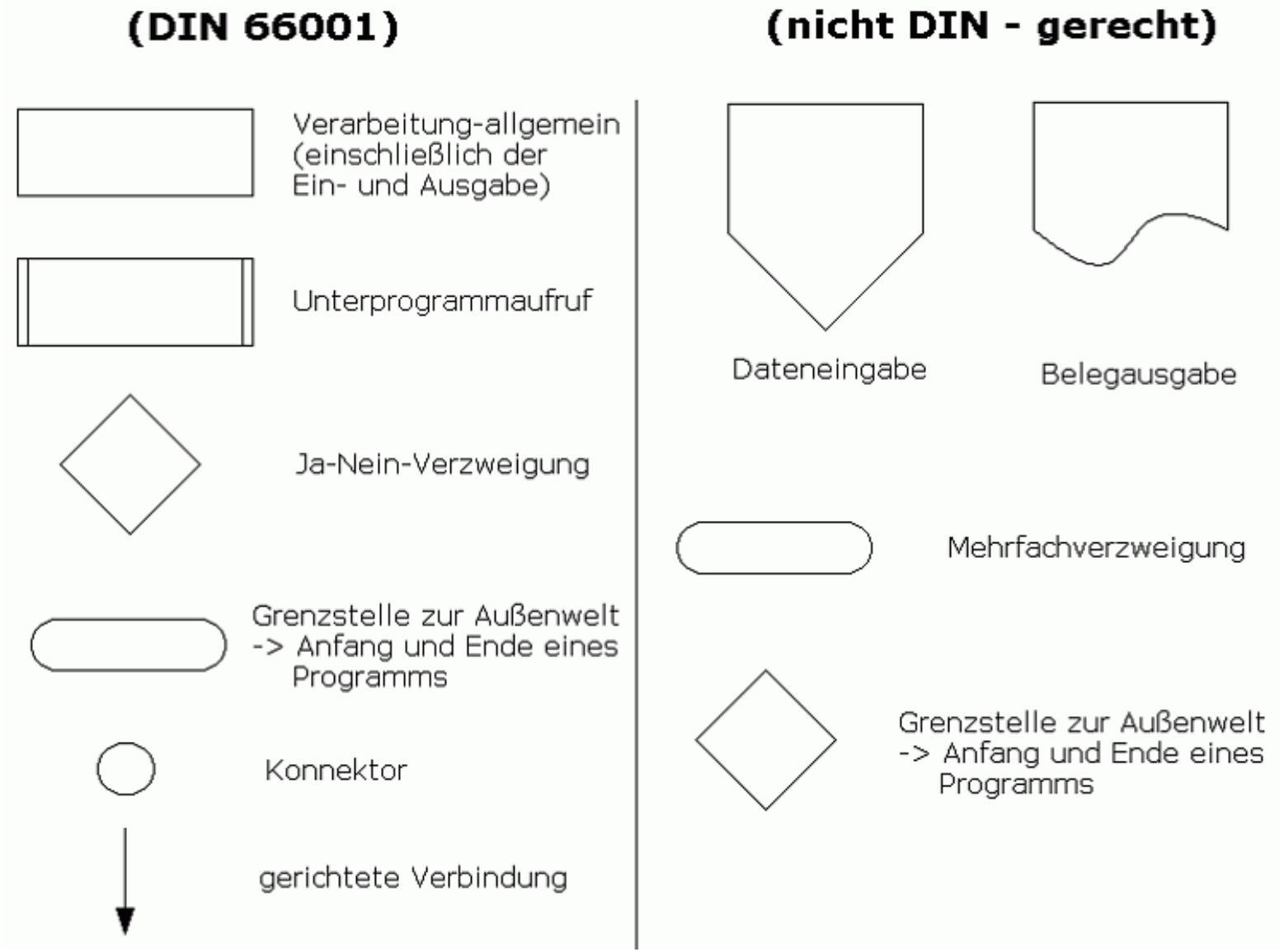
Verbale Beschreibung am Beispiel Geschwindigkeitsberechnung

- (Eingabeaufforderung an Benutzer)
- Eingabe des zurückgelegten Weges (über die Tastatur)
- Prüfung des eingegebenen Weges (>0), falls nicht >0: Ausgabe einer entsprechenden Fehlermeldung, neue Eingabe des Weges
- Eingabe der benötigten Zeit (über die Tastatur)
- Prüfung der eingegebenen Zeit (>0), falls nicht >0: Ausgabe einer entsprechenden Fehlermeldung,neue Eingabe der Zeit
- Berechnung der Geschwindigkeit (v=s/t)
- Ausgabe der Geschwindigkeit auf dem Bildschirm

Grafische Darstellungsform

- Früher wurde ein Algorithmus in Form eines Programmablaufplans (PAP) dargestellt.
- Heute ist man dazu übergegangen, den Algorithmus als Struktogramm darzustellen (heißt auch „Nassi-Shneiderman-Diagramm“).
- Bei größeren Projekten werden ganze Szenarien „modelliert“ und zum Beispiel als „Unified Modeling Language (UML)“ grafisch dargestellt, spezifiziert und dokumentiert.

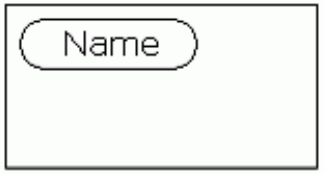
Programmablaufplan-Symbole



Struktogramm-Symbole

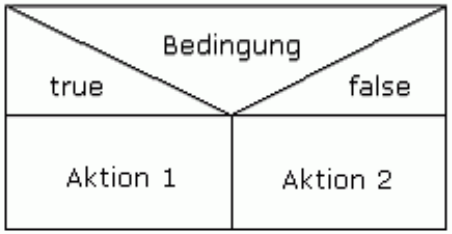
(DIN 66261)

Einfacher Strukturblock

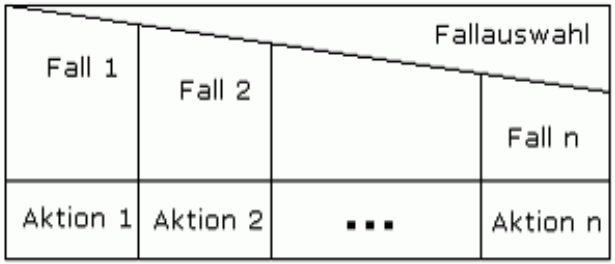


(nicht in der DIN enthalten)

Unterprogrammaufruf



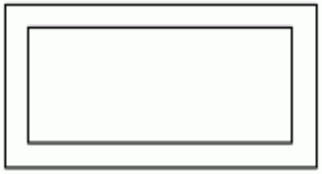
Auswahl



Mehrfachauswahl

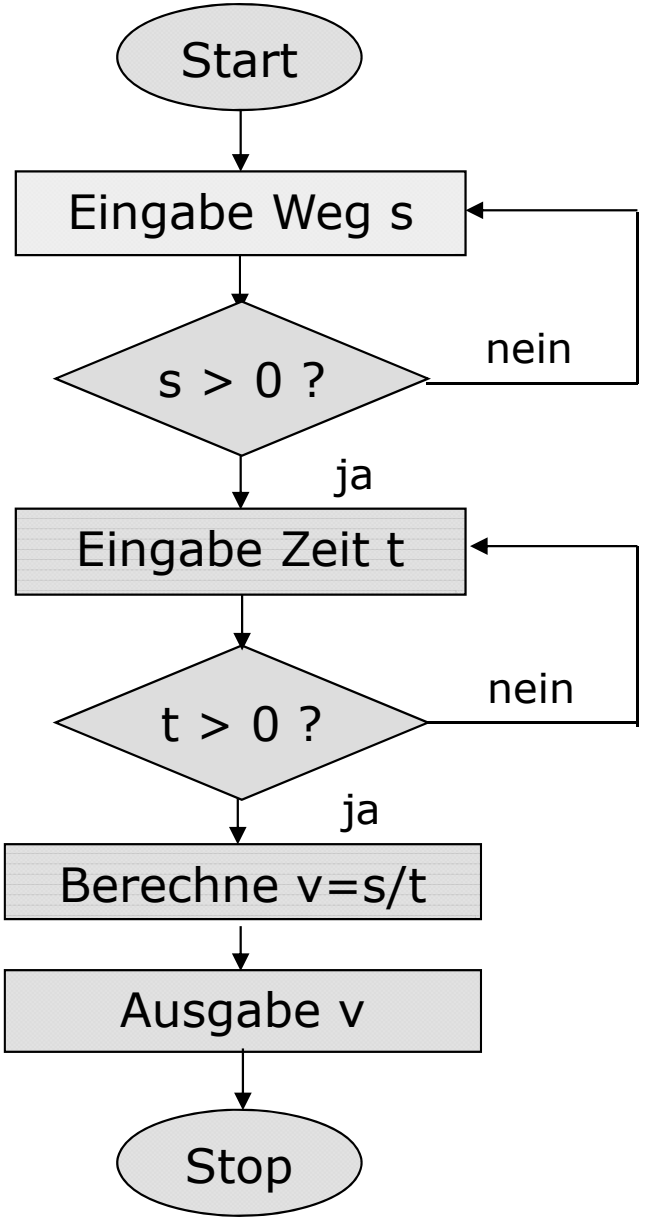
Schleife:
Abbruchbedingung am Beginn

Schleife:
Abbruchbedingung am Ende

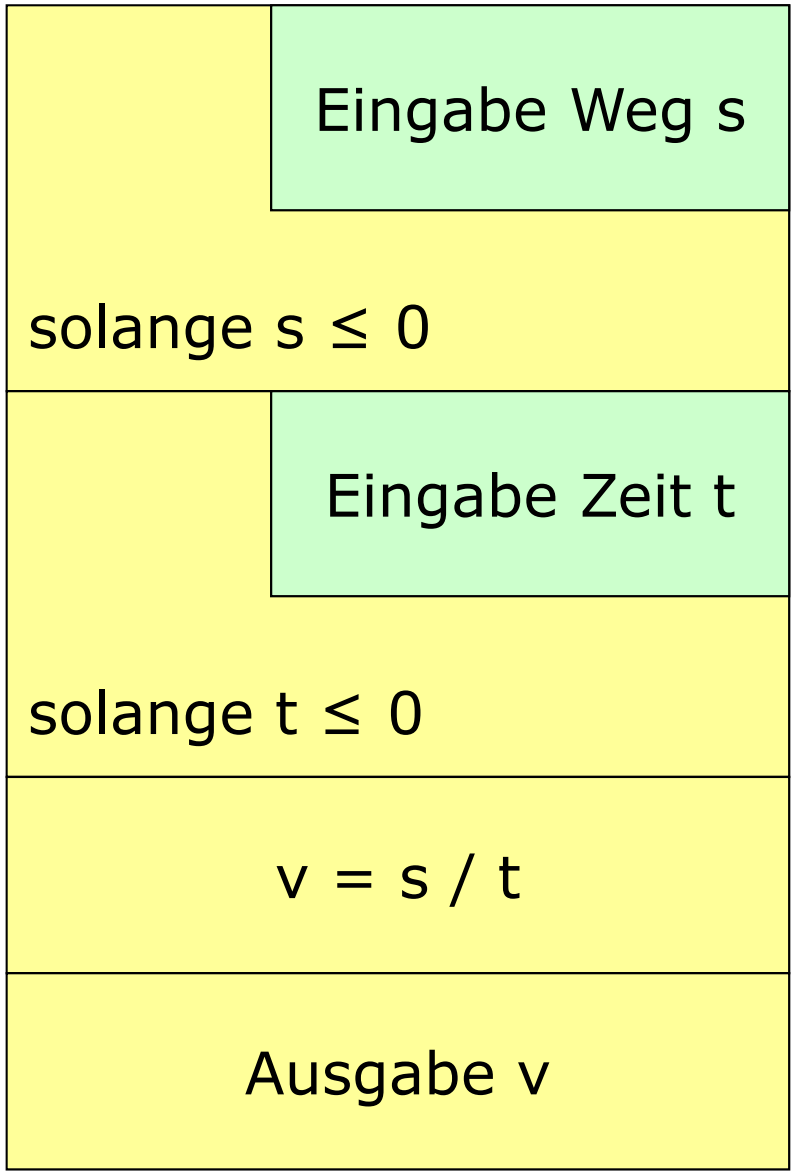


Block (Programmkennzeichnung)

Beispiel-Programmablaufplan



Beispiel-Struktogramm



Codieren und Testen

- Erstellen des Quellprogramms (Codieren)
  - Der Algorithmus wird in für den Computer verständliche Anweisungen einer Programmiersprache umgesetzt. Dies ergibt den sogenannten Quelltext oder Quellcode.
  - Dieser Quelltext wird dann durch den Compiler in Maschinenanweisungen übersetzt; der Linker „baut“ ein lauffähiges Programm.
- Testen des Programms
  - Für den Test des Programms wird es gestartet, d.h. in den Hauptspeicher geladen und vom Prozessor ausgeführt.
  - Neu entwickelte Programme weisen erfahrungsgemäß mehrere Fehler auf; nachdem die Fehler behoben wurden, muss das Programm ggf. auch mehrfach getestet werden.

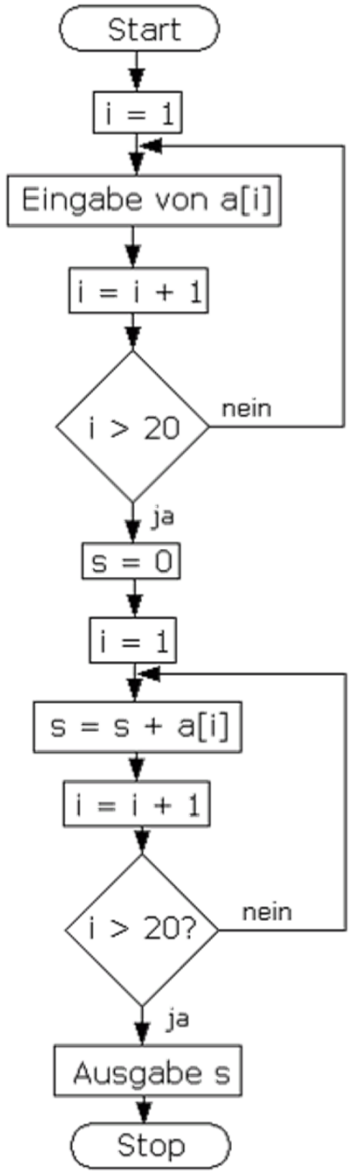
Übung: PAP / Struktogramm

Aufgabe:

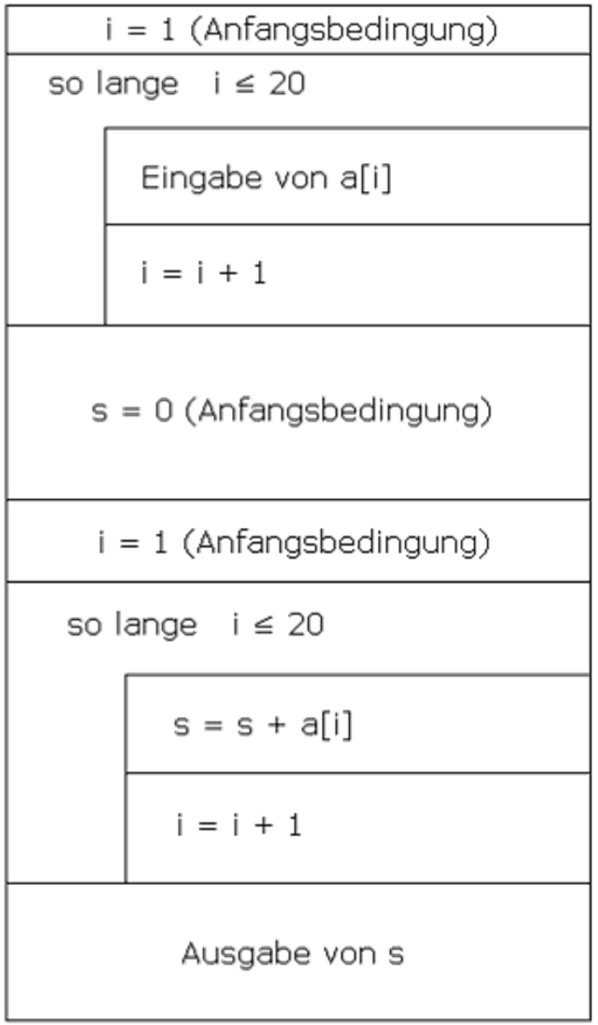
Der Benutzer möge 20 Ganzzahlen eingeben, anschließend wird die Summe aller eingegebenen Zahlen errechnet und ausgegeben

Mögliche Lösungen

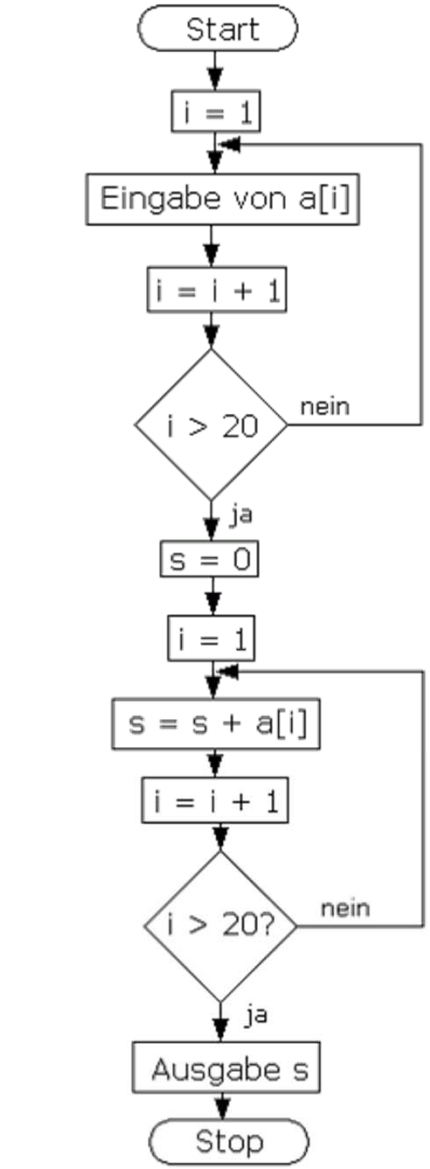
Programmablaufplan



Struktogramm



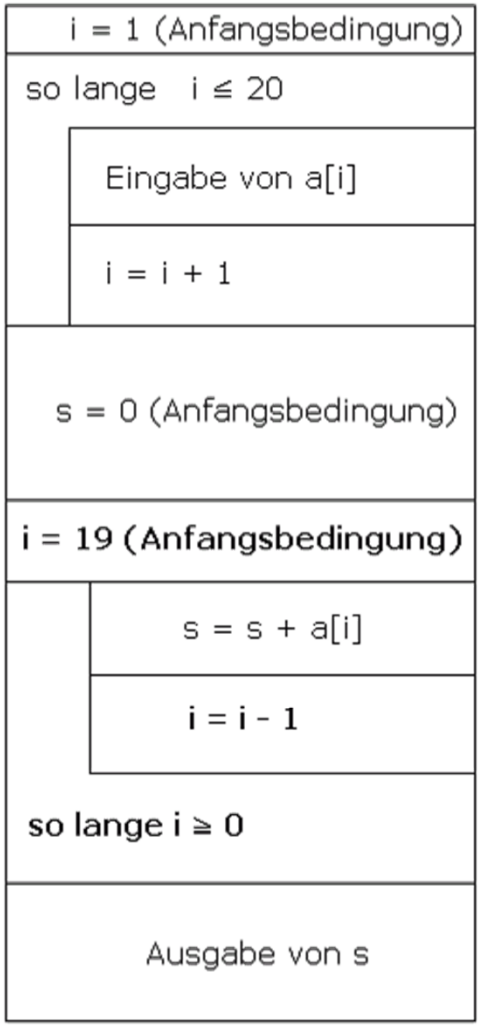
Möglicher Programmablaufplan




```
// Programmierbeispiel
#include <iostream.h>                          // Präprozessor


int main()                                     // Beginn des Programms
      {
            cout << "Beginn main()-Beispiel 05\n";
            int a[20], S, i;                   // Vereinbarungen

            cout << "Geben Sie zwanzig Zahlen ein:\n";



            for (i = 0; i < 20; i++)           // for-Schleife
            cin >> a[i];                       // Eingabe




            S = 0;                             // Anfangsbedingung
            i = 19;



            do                                 // do-while-Schleife
            {
                  S = S + a[i];                // Verarbeitung
                  i--;
            }     while (i >= 0);



            cout << "S[20] = " << S;           // Ausgabe

            cout << "\nEnde main()-Beispiel 05\n\n";
      return 0;                                // Ende des Programms
      }
```

Mögliches Struktogramm

Struktogramm




```
// Programmierbeispiel
#include <iostream.h>                    // Präprozessor

int main()                               // Beginn des Programms
      {
            cout << "Beginn main()-Beispiel 05\n";
            int a[20], S, i;             // Vereinbarungen

            cout << "Geben Sie zwanzig Zahlen ein:\n";


            for (i = 0; i < 20; i++)     // for-Schleife
            cin >> a[i];                 // Eingabe


            S = 0;                       // Anfangsbedingung
            i = 19;


            do                           // do-while-Schleife
            {
                  S = S + a[i];          // Verarbeitung
                  i--;
            }     while (i >= 0);


            cout << "S[20] = " << S;     // Ausgabe

            cout << "\nEnde main()-Beispiel 05\n\n";

      return 0;                          // Ende des Programms
      }
```

Programmbeispiel - Listing


```
// Programmierbeispiel für PAP und Struktogramm

# include <iostream>                          // Präprozessoranweisung

using namespace std;                          // Standardzuweisung

main()                                        //Beginn des Programms
{
   cout << "Beginn main()-Beispiel 05\n";     // Textausgabe
   int a[20], S, i;                           // Vereinbarungen

   cout << "Geben Sie zwanzig Zahlen ein:\n"; //Textausgabe

   for (i=0; i<20; i++)                       // for-Schleife
   cin >> a[i];                               // Zahleneingabe

   S=0;                                       // Anfangsbedingung s
   i=19;                                      // Anfangsbedingung i

   do                                         // do-while-Schleife
    {
      S=S+a[i];                               // Verarbeitung
      i--;
    } while (i>=0);                           // Abbruchkriterium

    cout << "\nEndsumme= " << S;              // Ausgabe des Ergebnisses

    cout << "\nEnde main()-Beispiel 05\n\n";  // Textausgabe

    getchar();                                // Anzeige alpha-numerische
    getchar();                                // Oberfläche (zwei Zeichen)

    return 0;                                 // Ende des Programms
}
```

Programmbeispiel - Laufzeit