Algorithmierung

Def: Deterministischer Algorithmus

Ein deterministischer Algorithmus ist eine Funktion f: $D_{ein} \longrightarrow D_{aus}$, die Eingabedaten oder Eingangsnachrichten D_{ein} in Ausgabedaten oder Ausgangsnachrichten D_{aus} schrittweise transformiert und dabei folgende Bedingungen erfüllt:

Exaktheit:	Die Funktion f kann präzise auf formale Weise beschrieben werden.
Finitheit:	Die Beschreibung von f ist endlich lang.
Vollständigkeit:	Die Beschreibung von f ist vollständig, d.h. sie umfasst auch alle Einzel/Sonderfälle (bzgl. der Ausgangsdaten)
Effektivität:	Die Einzelschritte von f sind elementar und real ausführbar.
Terminierung:	Die Funktion f hält nach endlich vielen Schritten an und liefert ein Resultat (d.h. eine Haltebedingung wird erreicht).
Determinismus:	Die Funktion liefert für die gleichen Eingabewerte stets das gleiche Ergebnis, wobei die Folge der Einzelschritte für jeden Eingabewert genau festgelegt ist.

Einzelschritte eines Algorithmus sind

a) Rechenanweisungen

b) Steueranweisungen

Sequenzen

Selektion (Fallunterscheidungen)

Iteration (Wiederholungen)

Rekursionen (Selbstaufrufende Wiederholungen)

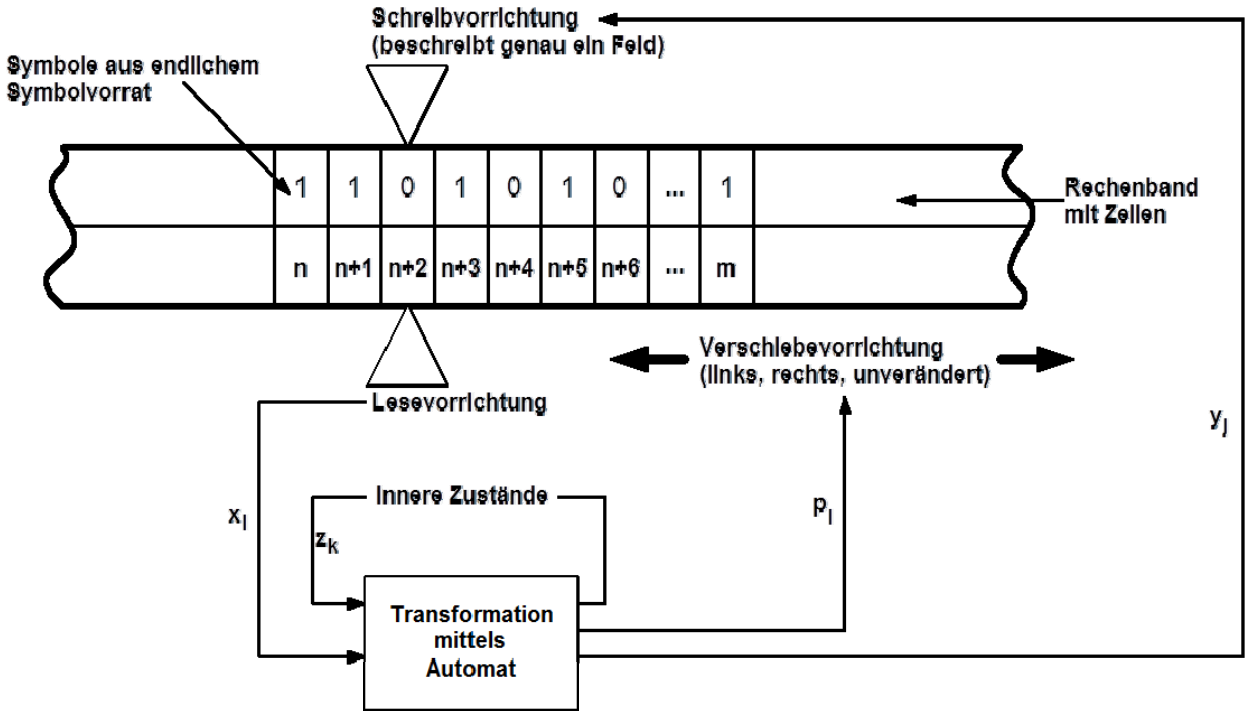
= algorithmische Grundstrukturen

Def: (Programm)

Ein Programm ist eine Beschreibung eines Algorithmus und seiner von ihm bearbeiteten Daten in einer Programmiersprache (Implementation).

Turing Maschine

(Alan Turing 1936)



- z_k = innere Zustände
- x_i = Eingabegröße
- y_j = Ausgabegröße
- p_i = Positionierungssteuerung

Turing Maschine

(Alan Turing 1936)

Die **Überführungsfunktion** läßt sich beschreiben mittels:

a) Funktionstabelle (Zustand, gelesenes Zeichen, Folgezustand, geschriebenes Zeichen, Kopfbewegung)

b) Zustandsgraph (Zustände als Knoten, gerichtete Bögen zwischen den Zuständen mit Label: gelesenes Zeichen, geschriebenes Zeichen, Kopfbewegung)

c) Programmiersprache mit den Befehlen:

IF char

WRITE(char)

LEFT

RIGHT

JUMP(Folgezustand)

HALT

Algorithmen - Theorie

Berechenbarkeit
Ein Algorithmus ist eine endliche Beschreibung über einem endlichen Alphabet.
Es existieren daher nur abzählbar viele Algorithmen, d.h. es existieren
nur abzählbar viele berechenbare Funktionen.

Aber, die Menge aller Funktionen f: N → N (mit N=Menge der natürlichen Zahlen) ist überabzählbar. Also existieren Funktionen, die sich nicht mittels eines Algorithmusses berechnen lassen. Also ist die Menge der definierbaren Funktionen umfangreicher, als die Menge der berechenbaren Funktionen.

Church-These 1936 (bis heute noch nicht bewiesen)
Jede berechenbare Funktion ist Turing-berechenbar.
Kein Berechnungsverfahren kann algorithmisch genannt
werden, das nicht von einer Turingmaschine ausführbar ist.
Jedes Problem, das maschinell lösbar ist, kann von einer Turingmaschine gelöst werden.

Dies bedeutet, dass alle Klassen von berechenbaren Funktionen äquivalent sind und es kann für jede dieser Klassen eine Turingmaschine konstruiert werden, welche die entsprechende Funktion berechnet. Die Menge der Turing-berechenbaren Fuktionen ist die Menge aller Funktionen, die sich mit einer Turingmaschine berechnen lassen.

Aus dieser These folgt die **Präzisierung des Algorithmenbegriffs:**
Ein Algorithmus ist eine Turing-berechenbare Funktion.
Eine Turingmaschine ist das formale Modell, um Algorithmen zu beschreiben.

Algorithmen - Theorie

Halteproblem:
Gibt es einen Algorithmus, mit dem entschieden werden kann, ob ein gegebenes Programm oder eine Maschine für alle Eingabedaten jemals die Haltebedingung erfüllt (und damit terminiert)? Für spezielle Programme kann dieser Nachweis evtl. geführt werden (aber eben nicht für ein beliebiges Programm) ⟶ Programmverifikation/Prüfen auf Korrektheit

Universelles Halteproblem:
Gibt es einen Algorithmus, mit dem entschieden werden kann, ob ein beliebiges Programm oder eine beliebige Maschine für alle Eingabedaten jemals die Haltebedingung erreicht (und damit terminiert)? ⟶ Antwort: **NEIN**

Entscheidbarkeit
Ein Problem oder eine Fragestellung für eine Menge oder Klasse M heißt entscheidbar, wenn es einen Algorithmus gibt, mit dem für jedes Element m aus M in endlicher Zeit festgestellt werden kann, ob für m die Problem-Frage eindeutig mit "JA" oder "NEIN" beantwortet werden kann.

Entscheidbare Probleme:
- Nachweise der syntaktisches Korrektheit einer Formel
- Nachweis der Allgemeingültigkeit eines aussagenlogischen Ausdrucks (Tautologie)

Nicht entscheidbare Probleme:
- PCP (Postsche Korrespondenzproblem)
- Halteproblem

Äquivalenzproblem
Erzeugen zwei Programme für jede beliebige Eingabe die jeweils gleiche Ausgabe?

Automatisierungswürdigkeit, d.h. wann ist die Programmierung sinnvoll:

a) komplizierter Lösungsalgorithmus

b) große Anzahl von Variablen

c) geforderte irrtumsfreie Berechnung/Entscheidung

d) Verarbeitungskapazität

e) geforderte hohe Geschwindigkeit

f) geforderte hohe Genauigkeit

g) wiederholbare Berechnungen (Iterationen)

h) schneller Zugriff zu Daten

Grundlegende Algorithmen-Methodiken/Strategien

a) Ausprobieren aller Möglichkeiten: Vollst. Enumeration / Methode der rohen Kraft (brute force method)

b) Teile und Herrsche ("devide and conquer")

c) Branch and Bound

d) Greedy (dt. gierig) Verfahren / Heuristiken (math. Optimierung)

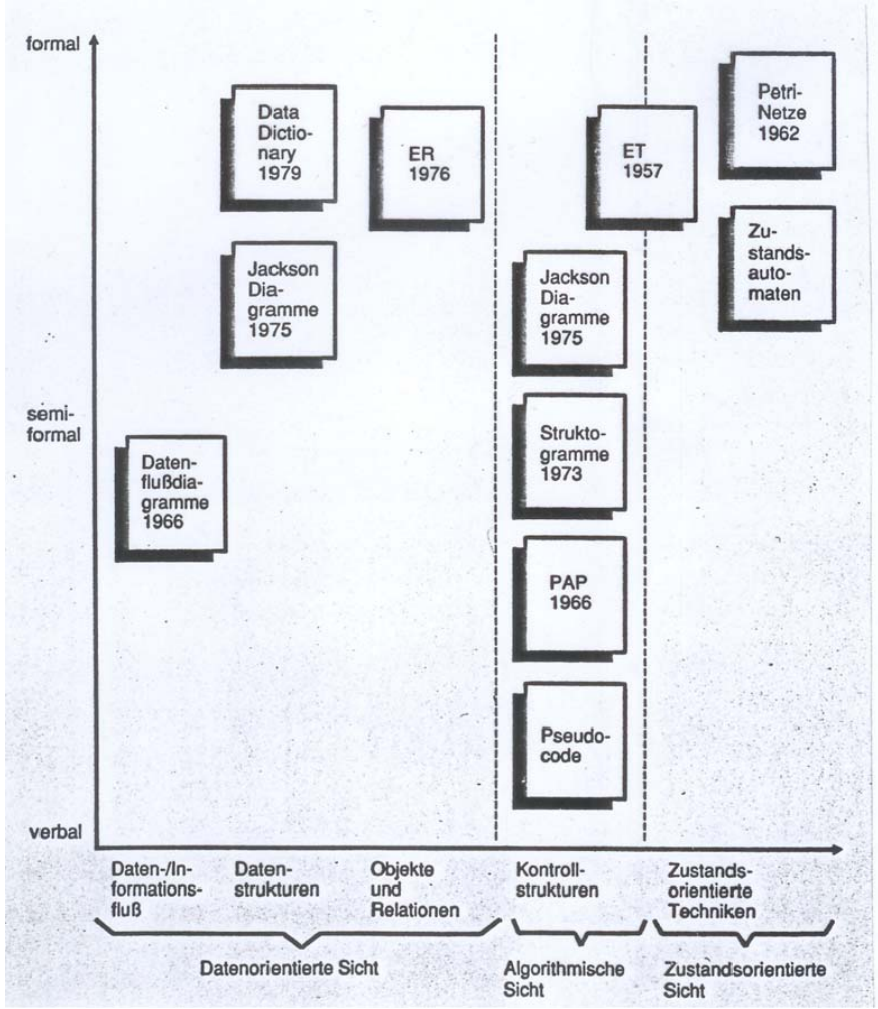
e) Versuch-und-Irrtum (trial and error)

f) Backtracking

g) löse-eine-vereinfachte-Version-zuerst-Strategien

Algorithmische Grundstrukturen und ihre Darstellung mittels

- Pseudocode
- Struktogramm
- PAP
- Programmlinienmethode



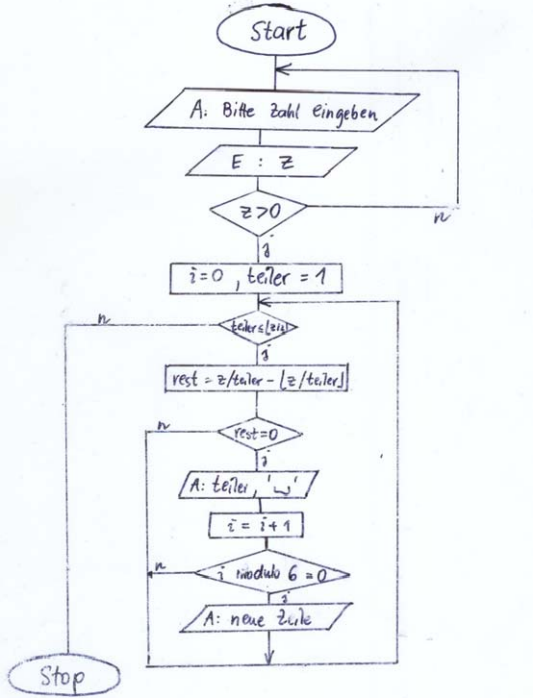
Pseudocode

(Bestimmung aller Teiler einer ganzen Zahl)

```
K: Bestimmung aller Teiler einer ganzen Zahl
Führe aus:
    A: 'Bitte Zahl eingeben!'
    E: z
Solange bis    z>0
Setze:  i=0 , teiler = 1
Solange  teiler ≤ ⌊z/2⌋  führe aus:
    Setze:  rest = z/teiler - ⌊z/teiler⌋
    Wenn  rest = 0  dann
        A: teiler, '␣'
        Setze: i = i+1
        Wenn  i modulo 6 = 0  dann
            A: neue Zeile
        Sonst
    Sonst
    Setze: teiler = teiler + 1
Ende
```

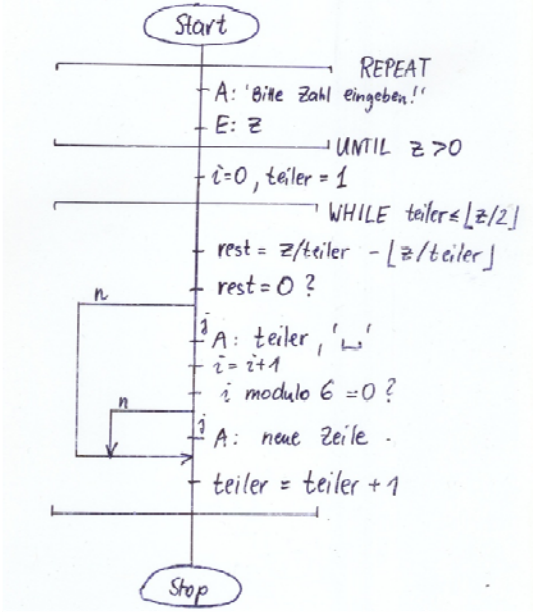
PAP mit Sinnbildern

(Bestimmung aller Teiler einer ganzen Zahl)



Programmlinienmethode

(Bestimmung aller Teiler einer ganzen Zahl)



Unified Modeling Language (UML)

- Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme (graphische Modellierungssprache)
- Standardisierung durch die Object Management Group(OMG) (1999) und ISO/IEC 19501 (Version 2.1.2)
- Seit UML2 Splittung in Infrastructure Specification (Klassen, Assoziation, Multiplizität), Superstructure Specification (Modellelemente:Anwendungsfall, Aktivität, Zustandsautomat), Object Constraint Language, Diagram Interchange
- 4-Ebenen-Hierarchie einer Metamodellierung: M3 (Metametamodell-Meta Object Facility (MOF)),M2 (Metamodell-UML),M1 (Benutzermodell), M0 (Objekte als Laufzeitinstanzen)
- Austausch von Modellen und Diagrammen mittels XML Metadata Interchange (XMI)
- UML2.3 Strukturdiagramme: Klassendiagramm, Kompositionsstrukturdiagramm, Komponentendiagramm, Verteilungsdiagramm, Objektdiagramm, Paketdiagramm, Profildiagramm
- UML2.3 Verhaltensdiagramme: Aktivitätsdiagramm, Anwendungsfalldiagramm/Use-Case/Nutzfalldiagramm, Interaktionsübersichtsdiagramm, Kommunikationsdiagramm, Sequenzdiagramm, Zeitverlaufsdiagramm, Zustandsdiagramm

Algorithmische Grundstrukturen
(Sequenz, Alternative, Sprung, Zyklen, I/O, UP, Fortsetzungsmarke)

Anweisung	=	einfache Programmaktion
Sequenz	=	Folge von Anweisungen, logischen Elementarstrukturen oder komplexen Programmstrukturen
Alternative	=	Verzweigung bzgl. des Erfülltseins/Nichterfülltseins einer Bedingung/Entscheidungsfrage mit den zwei Antworten Ja/Nein
Sprung	=	bedingte/unbedingte Fortsetzung des Programmablaufes an einer anderen Stelle
Zyklus	=	wiederholte Ausführung von Programmteilen bis eine Abbruchbedingung erfüllt ist

Bsp.1: Lösung der allgemeinen quadratischen Gleichung

suche alle Lösg. der Gl.: $ax^2 + bx + c = 0$

IN: a, b, c

OUT: x1, x2

Fall: $a = 0 \Rightarrow bx + c = 0$

Fall: $b = 0 \Rightarrow c = 0$

Fall: $c = 0 \Rightarrow$ OUT: jede Zahl x ist Lösg.

$c \neq 0$ aber $c = 0$ ↯ OUT: es ex. kein Lösg. x

$b \neq 0 \Rightarrow bx + c = 0$

$x = -c/b$

OUT: "Lösg. x1 = ", x, "

x2 = ", x

Fall $a \neq 0 \Rightarrow x^2 + \frac{b}{a}x + \frac{c}{a} = 0$

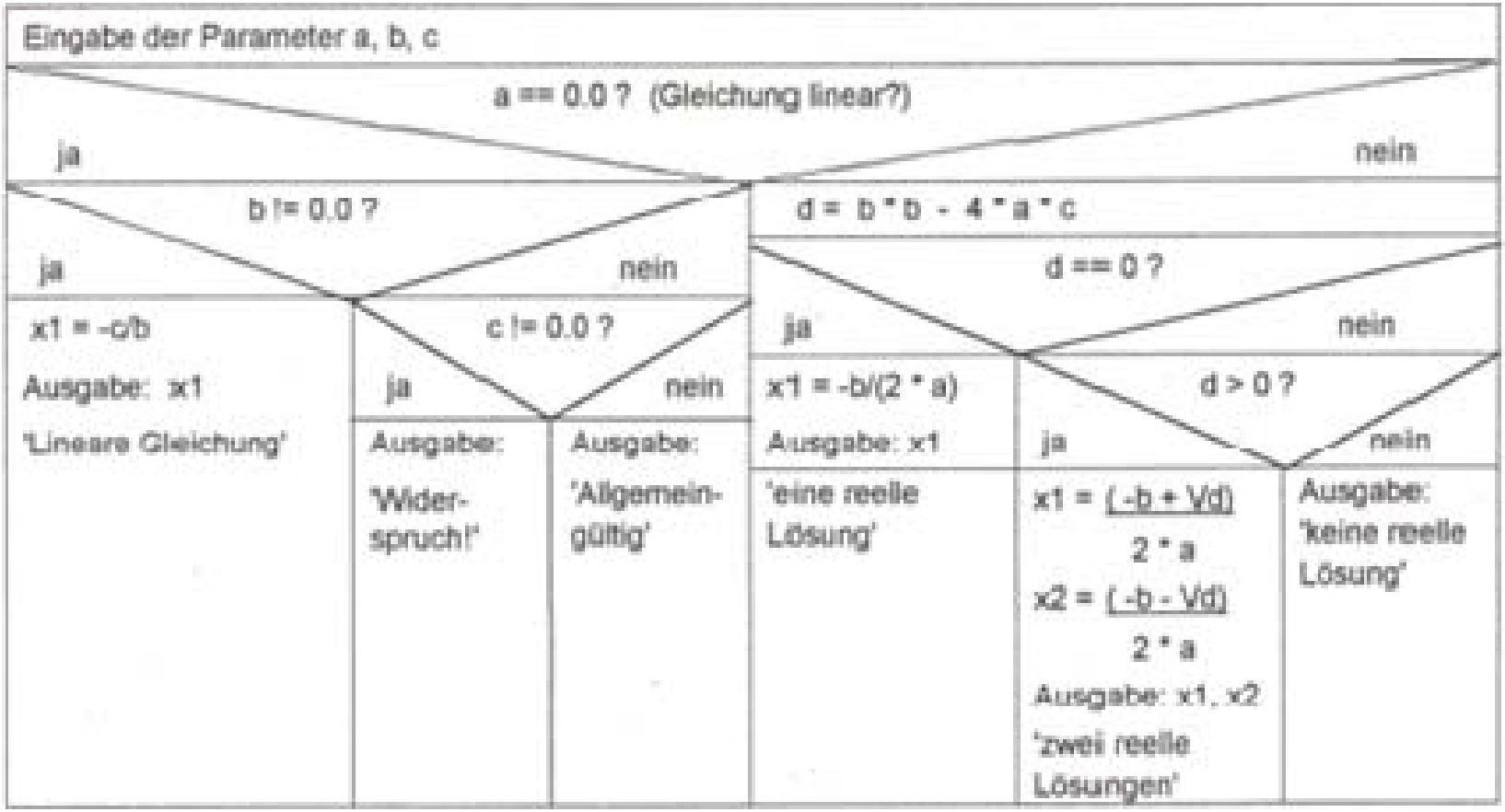
$p = \frac{b}{a}$, $q = \frac{c}{a}$

$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$ $\quad D = \frac{p^2}{4} - q$

Fall: $D \geq 0 \Rightarrow w = \sqrt{D} \Rightarrow x1 = -\frac{p}{2} + w,\ x2 = -\frac{p}{2} - w$

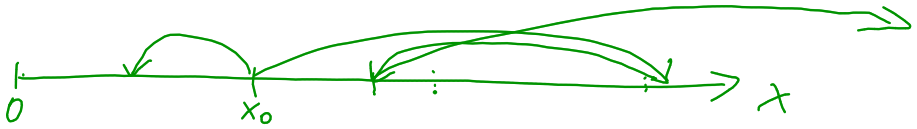
$D < 0 \Rightarrow w = \sqrt{|D|} \Rightarrow x1 = -\frac{p}{2} + i \cdot w,\ x2 = -\frac{p}{2} - i \cdot w$

OUT: x1, x2


Eingabe der Parameter a, b, c
a == 0.0 ? (Gleichung linear?)
ja
nein
b != 0.0 ?
ja
nein
x1 = -c/b
Ausgabe: x1
'Lineare Gleichung'
c != 0.0 ?
ja
nein
Ausgabe:
'Wider-
spruch!'
Ausgabe:
'Allgemein-
gültig'
d = b * b - 4 * a * c
d == 0 ?
ja
nein
x1 = -b/(2 * a)
Ausgabe: x1
'eine reelle
Lösung'
d > 0 ?
ja
nein
x1 = (-b + Vd)
2 * a
x2 = (-b - Vd)
2 * a
Ausgabe: x1, x2
'zwei reelle
Lösungen'
Ausgabe:
'keine reelle
Lösung'

Bsp.2: Collatz Problem

$$x_{i+1} = \begin{cases} \frac{x_i}{2} & \text{falls } x_i \text{ gerade} \\ 3x_i + 1 & \text{falls } x_i \text{ ungerade} \end{cases}$$




```
FUNCTION (x)
{ WHILE (x >1) DO
    { IF (x mod 2) ==0 THEN x = x/2;
      ELSE x = 3·x + 1;
    }
  RETURN x
}
```

x = 1
x = 2 → 1
x = 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1
x = 4
x = 5
x = 6 → 3
x = 7 → 22 → 11 → 34 → 17 → 52 → 26 → 13 → 40 → 20 → 10 → 5

x=1 #0#
x=2 1 #0#
x=3 10 5 16 8 4 2 1 #16#
x=4 2 1 #16#
x=5 16 8 4 2 1 #16#
x=6 3 10 5 16 8 4 2 1 #16#
x=7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 #16#
x=8 4 2 1 #16#
x=9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 #16#
...
x=27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 #16#
x=29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 #16#
x=30 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308
...
x=5460 2730 1365 4096 2048 1024 512 256 128 64 32 16 8 4 2 1 **#4096#**
...
x=6170 3085 9256 4628 2314 1157 3472 1736 868 434 217 652 326 163 490 245 736 368 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=6171 18514 9257 27772 13886 6943 20830 10415 31246 15623 46870 23435 70306 35153 105460 52730 26365 79096 39548 19774 9887 29662 14831 44494 22247 66742 33371 100114 50057 150172 75086 37543 112630 56315 168946 84473 253420 126710 63355 190066 95033 285100 142550 71275 213826 106913 320740 160370 80185 240556 120278 60139 180418 90209 270628 135314 67657 202972 101486 50743 152230 76115 228346 114173 342520 171260 85630 42815 128446 64223 192670 96335 289006 144503 433510 216755 650266 325133 975400 487700 243850 121925 365776 182888 91444 45722 22861 68584 34292 17146 8573 25720 12860 6430 3215 9646 4823 14470 7235 21706 10853 32560 16280 8140 4070 2035 6106 3053 9160 4580 2290 1145 3436 1718 859 2578 1289 3868 1934 967 2902 1451 4354 2177 6532 3266 1633 4900 2450 1225 3676 1838 919 2758 1379 4138 2069 6208 3104 1552 776 388 194 97 292 146 73 220 110 55 166 83 250 125 376 188 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=6172 3086 1543 4630 2315 6946 3473 10420 5210 2605 7816 3908 1954 977 2932 1466 733 2200 1100 550 275 826 413 1240 620 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=6173 18520 9260 4630 2315 6946 3473 10420 5210 2605 7816 3908 1954 977 2932 1466 733 2200 1100 550 275 826 413 1240 620 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#

x=9999 29998 14999 44998 22499 67498 33749 101248 50624 25312 12656 6328 3164 1582 791 2374 1187 3562 1781 5344 2672 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 #16#
x=10000 5000 2500 1250 625 1876 938 469 1408 704 352 176 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 #16#
maximale Iterationslänge bis 10000 =261 bei Zahl=6171

Bsp.3: Bestimmung des größten gemeinsamen Teilers zweier ganzer Zahlen a, b

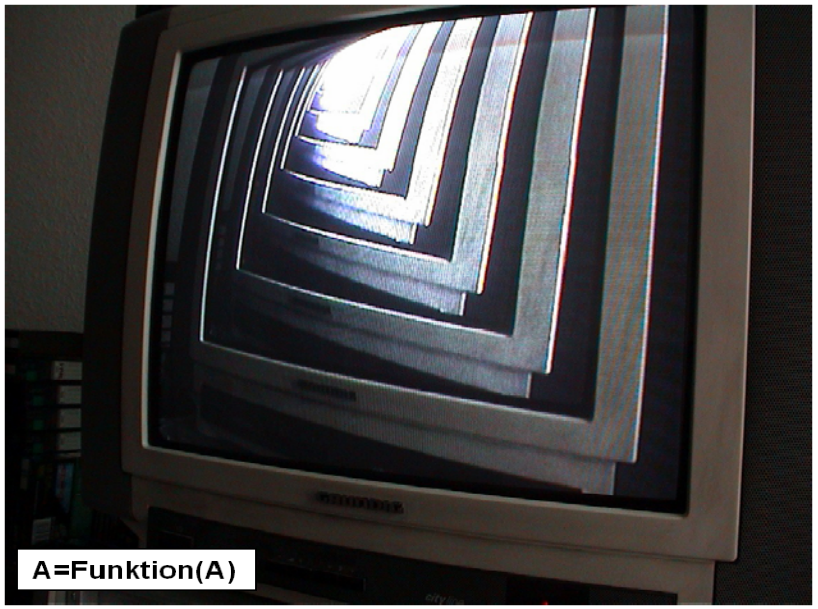
Lösg 1: a p_1 p_2 p_3 p_4 p_5 p_6 p_7
b p_2 p_4
$p_2 \cdot p_4$ $ggT = p_2 \cdot p_4$

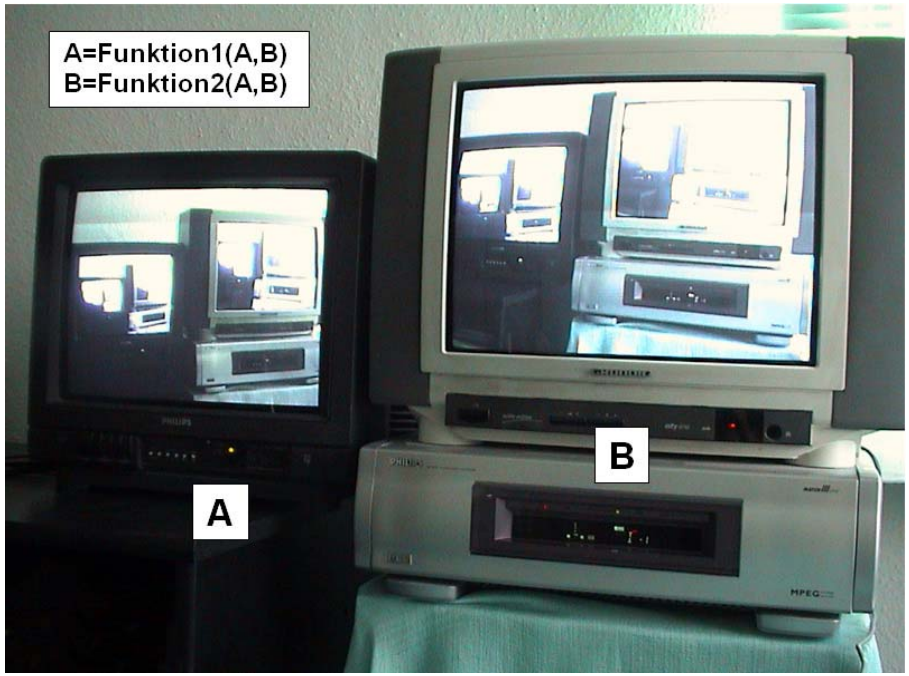
Lösg 2:
$$ggT(a,b) = \begin{cases} b & \text{falls } a \bmod b = 0 \text{ (d.h. } b \mid a \text{, } b \text{ teilt } a) \\ ggT(b,a) & \text{falls } b > a \\ ggT(a-b,b) & \text{sonst} \end{cases}$$

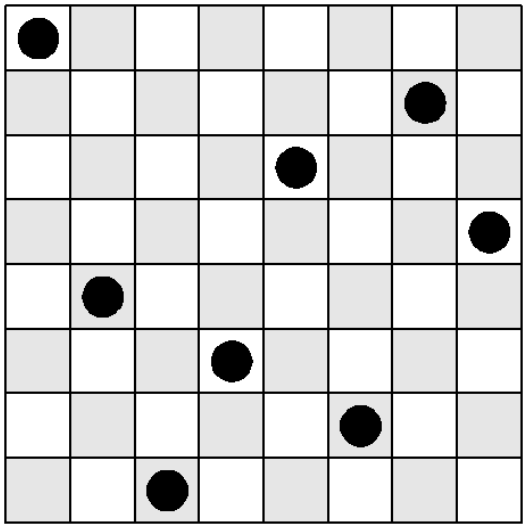
Lösg. 3: Euklidischer Algorithmus
$$EUKLID(a,b) = \begin{cases} b & \text{falls } a \bmod b = 0 \\ EUKLID(b,a) & \text{falls } a < b \\ EUKLID(b, \underbrace{a \bmod b}_{<b}) & \text{sonst} \end{cases}$$

Bsp: $ggT(4290, 2618) = ggT(1672, 2618) = ggT(2618, 1672) = \ldots$

$EUKLID(4290, 2618) = EUKLID(2618, 1672) = \ldots$







	4				7			
2			9				1	
					1		6	8
4		8		1			7	
			4		3			
	2			6		3		4
6	7		8					
	8				9			2
			5				8	

8	4	1	6	5	7	2	3	9
2	6	3	9	8	4	5	1	7
5	9	7	3	2	1	4	6	8
4	3	8	2	1	5	9	7	6
7	5	6	4	9	3	8	2	1
1	2	9	7	6	8	3	5	4
6	7	4	8	3	2	1	9	5
3	8	5	1	7	9	6	4	2
9	1	2	5	4	6	7	8	3

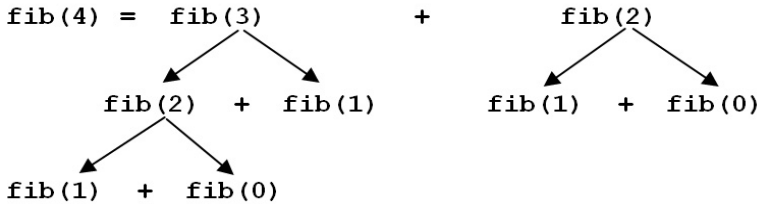
Bsp.4: Fibonacci-Folge (Rekursion)

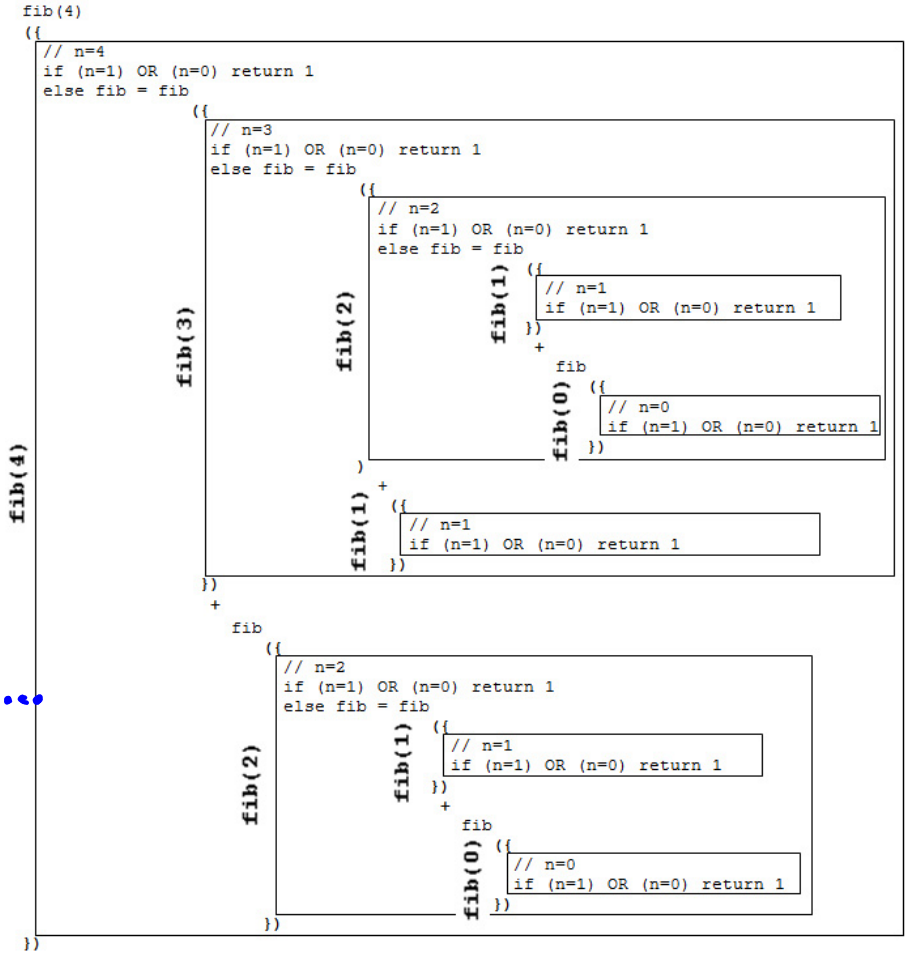
einfache Rekursion (Fakultät)

$f(n) = n! = n \cdot f(n-1) = n \cdot (n-1)!$

```
fib(n)                 log. Operator || == Vergleichsoperator
{ IF (n=1) OR (n=0) THEN RETURN 1
  ELSE {RETURN fib(n-1) + fib(n-2) }
}
```

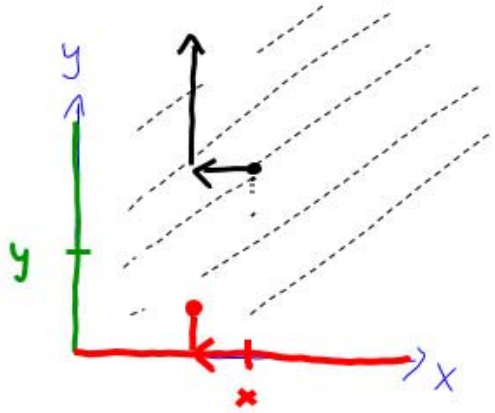
Fibonacci-Zahlen: 1,1,2,3,5,8,13,21,...





Bsp.5: Ackermann Funktion (nicht primitive Rekursion)

```
int   ack ( int x, int y)
  { if (x==0) return y+1
    else if (y==0) then return ack(x-1,1)
         else return ack (x-1, ack(x,y-1))
  }
```

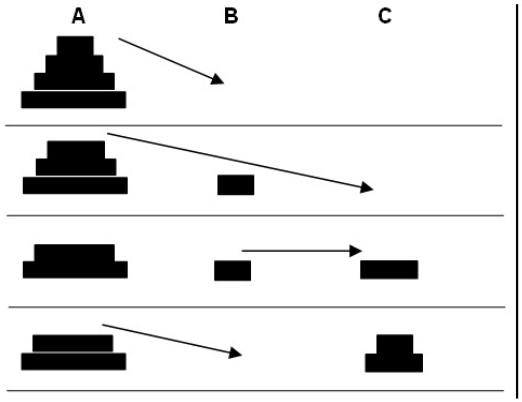


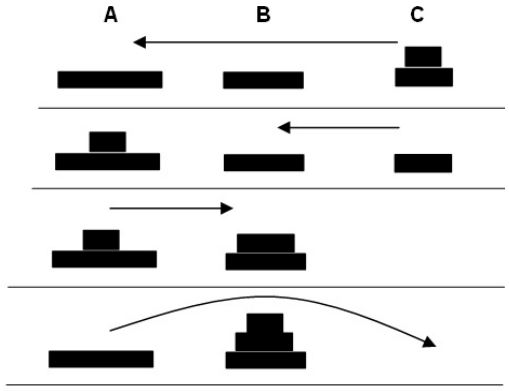
```
/* gcc _test1a.c -o _test1a.exe */
#include <stdio.h>
int ackanz;
int ack(int x, int y);
main()
{  int a;
   ackanz=0;a=ack(1,0);printf("Ackermann(1,0)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(0,1);printf("\nAckermann(0,1)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(1,1);printf("\nAckermann(1,1)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(1,2);printf("\nAckermann(1,2)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(2,1);printf("\nAckermann(2,1)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(2,2);printf("\nAckermann(2,2)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(2,3);printf("\nAckermann(2,3)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(3,2);printf("\nAckermann(3,2)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(3,3);printf("\nAckermann(3,3)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(3,4);printf("\nAckermann(3,4)=%i  mit %i rekursiven Aufrufen!", a, ackanz);
   ackanz=0;a=ack(4,3);printf("\nAckermann(4,3)=%i  mit %i rekursiven Aufrufen!", a,ackanz);
   /*ackanz=0;a=ack(4,4);printf("\nAckermann(4,4)=%i  mit %i rekursiven Aufrufen!", a,ackanz);*/
}
int ack(int x, int y)
{ ackanz++;
  if (ackanz % 10000000 ==0) printf("\nAufruf Nr. %i",ackanz);
  if (x==0) {return y+1;}
  else {if (y==0) {return ack(x-1,1);}
        else {return ack(x-1, ack(x,y-1));}}
}
```

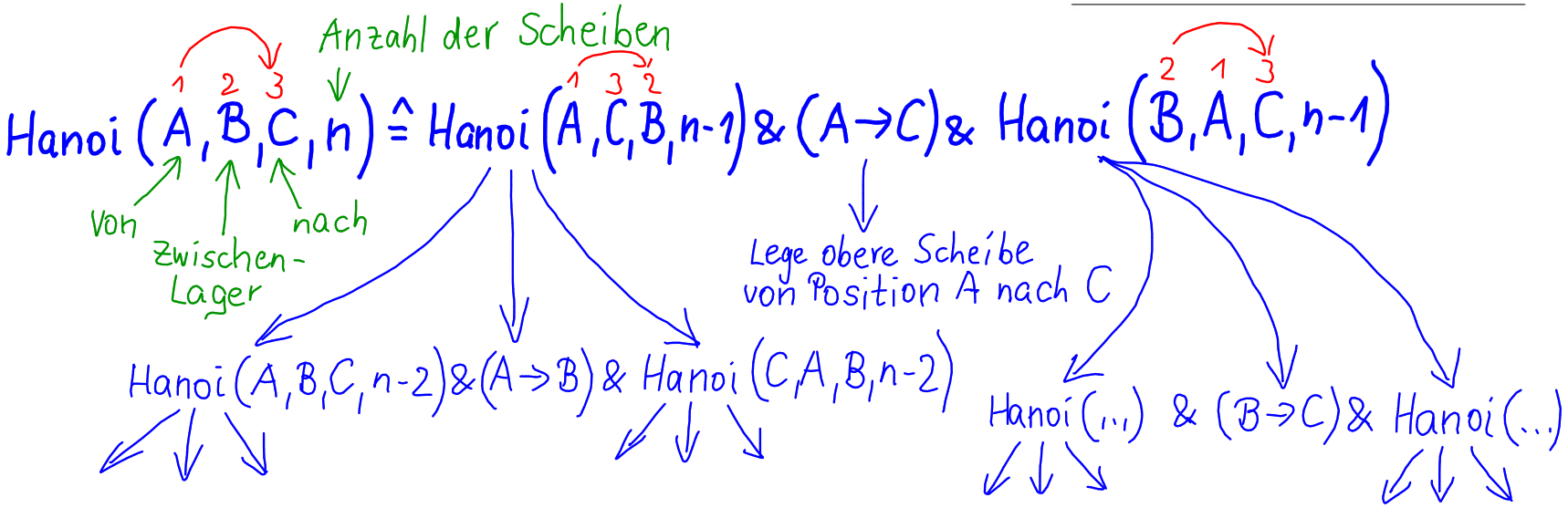
```
/* Ausgabe

C:\gcc-2.95.2\bin>_test1a.exe
Ackermann(1,0)=2  mit 2 rekursiven Aufrufen!
Ackermann(0,1)=2  mit 1 rekursiven Aufrufen!
Ackermann(1,1)=3  mit 4 rekursiven Aufrufen!
Ackermann(1,2)=4  mit 6 rekursiven Aufrufen!
Ackermann(2,1)=5  mit 14 rekursiven Aufrufen!
Ackermann(2,2)=7  mit 27 rekursiven Aufrufen!
Ackermann(2,3)=9  mit 44 rekursiven Aufrufen!
Ackermann(3,2)=29  mit 541 rekursiven Aufrufen!
Ackermann(3,3)=61  mit 2432 rekursiven Aufrufen!
Ackermann(3,4)=125  mit 10307 rekursiven Aufrufen!
Ackermann(4,3)=
Aufruf Nr. 10000000
Aufruf Nr. 20000000
Aufruf Nr. 30000000
Aufruf Nr. 40000000
Aufruf Nr. 50000000
Aufruf Nr. 60000000
Aufruf Nr. 70000000
Aufruf Nr. 80000000
Aufruf Nr. 90000000
Aufruf Nr. 100000000
Aufruf Nr. 110000000
Aufruf Nr. 120000000
Aufruf Nr. 130 000 000^C
C:\gcc-2.95.2\bin>         */
```

Bsp.6: Turm von Hanoi

Turm von Hanoi:
Lösung für (n+1) von A ➔ C =
Lösung für n von A ➔ B mit Zwischenlager C
(n+1)-te Scheibe von A nach C
Lösung für n von B ➔ C mit Hilfslager A







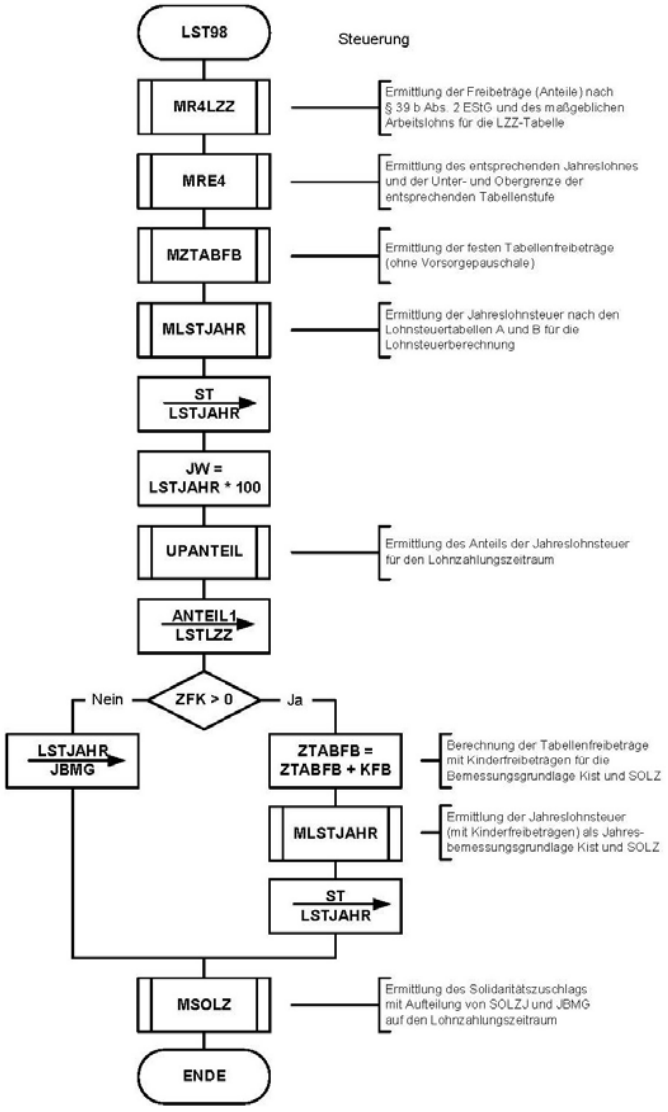
```
/* gcc _hanoi.c -o _hanoi.exe */
#include <stdio.h>

int anz;
int rekhanoi(int x,char a, char b, char c);

int main(void)
{
        int x;
        for (x=3;x<=10;x++)
             { anz=0;
               printf ("\nWieviel Scheiben : %i", x);
                rekhanoi(x,'A','B','C');printf("\n");
             }
}

int rekhanoi(int x, char a, char b, char c)
{
      if(x==1)
            {   anz++;
                printf ("\n%i.obere Scheibe von %c nach %c",anz,a,c);
                return;
            }
      rekhanoi(x-1,a,c,b);
      rekhanoi(1,a,b,c);
      rekhanoi(x-1,b,a,c);
}
```

```
Wieviel Scheiben : 3
1.obere Scheibe von A nach C
2.obere Scheibe von A nach B
3.obere Scheibe von C nach B
4.obere Scheibe von A nach C
5.obere Scheibe von B nach A
6.obere Scheibe von B nach C
7.obere Scheibe von A nach C

Wieviel Scheiben : 4
1.obere Scheibe von A nach B
2.obere Scheibe von A nach C
3.obere Scheibe von B nach C
4.obere Scheibe von A nach B
5.obere Scheibe von C nach A
6.obere Scheibe von C nach B
7.obere Scheibe von A nach B
8.obere Scheibe von A nach C
9.obere Scheibe von B nach C
10.obere Scheibe von B nach A
11.obere Scheibe von C nach A
12.obere Scheibe von B nach C
13.obere Scheibe von A nach B
14.obere Scheibe von A nach C
15.obere Scheibe von B nach C

Wieviel Scheiben : 5
1.obere Scheibe von A nach C
2.obere Scheibe von A nach B
3.obere Scheibe von C nach B
4.obere Scheibe von A nach C
5.obere Scheibe von B nach A
6.obere Scheibe von B nach C
7.obere Scheibe von A nach C
8.obere Scheibe von A nach B
9.obere Scheibe von C nach B
10.obere Scheibe von C nach A
11.obere Scheibe von B nach A
12.obere Scheibe von C nach B
13.obere Scheibe von A nach C
14.obere Scheibe von A nach B
15.obere Scheibe von C nach B
16.obere Scheibe von A nach C
17.obere Scheibe von B nach A
18.obere Scheibe von B nach C
19.obere Scheibe von A nach C
20.obere Scheibe von B nach A
21.obere Scheibe von C nach B
22.obere Scheibe von C nach A
23.obere Scheibe von B nach A
24.obere Scheibe von B nach C
25.obere Scheibe von A nach C
26.obere Scheibe von A nach B
27.obere Scheibe von C nach B
28.obere Scheibe von A nach C
29.obere Scheibe von B nach A
30.obere Scheibe von B nach C
31.obere Scheibe von A nach C

Wieviel Scheiben : 6
1.obere Scheibe von A nach B
2.obere Scheibe von A nach C
3.obere Scheibe von B nach C
4.obere Scheibe von A nach B
5.obere Scheibe von C nach A
6.obere Scheibe von C nach B
7.obere Scheibe von A nach B
8.obere Scheibe von A nach C
9.obere Scheibe von B nach C
10.obere Scheibe von B nach A
11.obere Scheibe von C nach A
12.obere Scheibe von B nach C
13.obere Scheibe von A nach B
14.obere Scheibe von A nach C
15.obere Scheibe von B nach C
16.obere Scheibe von A nach B
17.obere Scheibe von C nach A
18.obere Scheibe von C nach B
19.obere Scheibe von A nach B
20.obere Scheibe von C nach A
21.obere Scheibe von B nach C
22.obere Scheibe von B nach A
23.obere Scheibe von C nach A
24.obere Scheibe von C nach B
25.obere Scheibe von A nach B
26.obere Scheibe von A nach C
27.obere Scheibe von B nach C
28.obere Scheibe von A nach B
29.obere Scheibe von C nach A
30.obere Scheibe von C nach B
31.obere Scheibe von A nach B
32.obere Scheibe von A nach C
33.obere Scheibe von B nach C
34.obere Scheibe von B nach A
35.obere Scheibe von C nach A
36.obere Scheibe von B nach C
37.obere Scheibe von A nach B
38.obere Scheibe von A nach C
39.obere Scheibe von B nach C
40.obere Scheibe von B nach A
41.obere Scheibe von C nach A
42.obere Scheibe von C nach B
43.obere Scheibe von A nach B
44.obere Scheibe von C nach A
45.obere Scheibe von B nach C
46.obere Scheibe von B nach A
47.obere Scheibe von C nach A
48.obere Scheibe von B nach C
49.obere Scheibe von A nach B
50.obere Scheibe von A nach C
51.obere Scheibe von B nach C
52.obere Scheibe von A nach B
53.obere Scheibe von C nach A
54.obere Scheibe von C nach B
55.obere Scheibe von A nach B
56.obere Scheibe von A nach C
57.obere Scheibe von B nach C
58.obere Scheibe von B nach A
59.obere Scheibe von C nach A
60.obere Scheibe von B nach C
61.obere Scheibe von A nach B
62.obere Scheibe von A nach C
63.obere Scheibe von B nach C
```

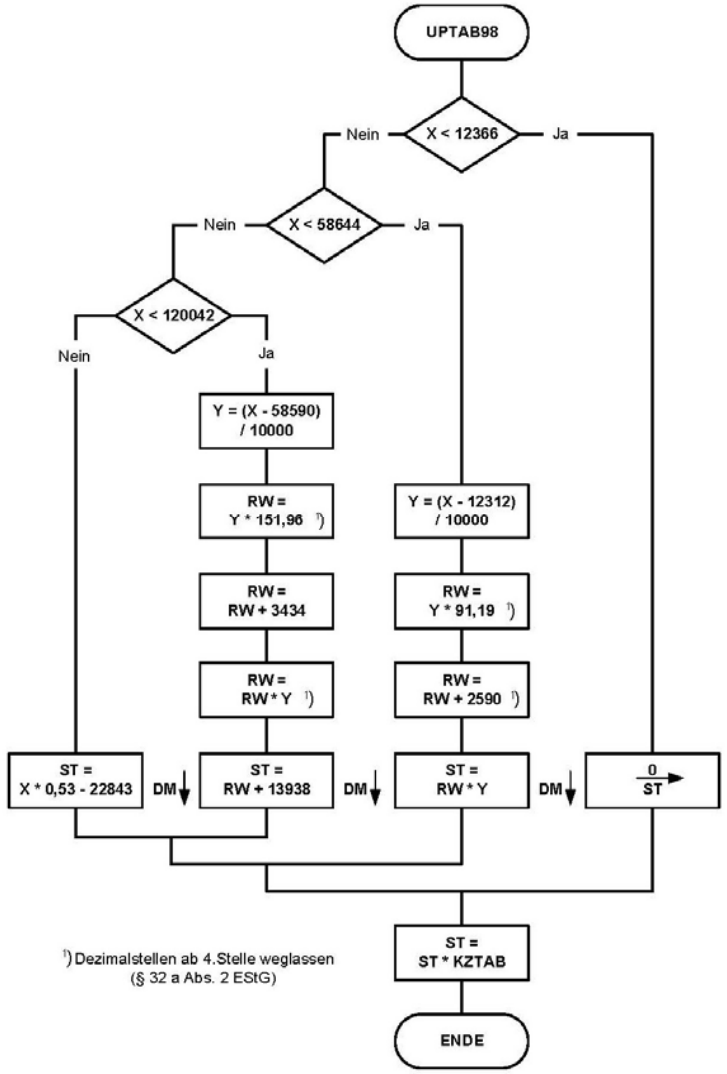
allgemein bei n-Scheiben: $2^n - 1$ Scheibentransfers

Bsp.7: Berechnung der Einkommenssteuer

Programmablaufplan EkSt 1998



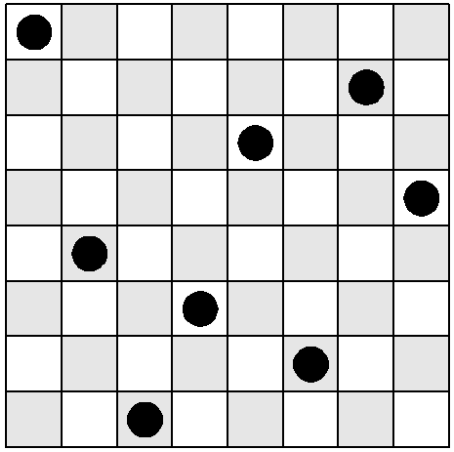
Tarifliche Einkommensteuer (§ 32 a EStG)



Bsp.8: 8-Damen-Problem

Ziel: Positioniere derart 8 Damen auf einem Schachbrett, dass keine zwei Damen sich gegenseitig „schlagen“ können

```
main()
{
int anz=0,g=0,i,j,s,z1,z2,z3,z4,z5,z6,z7,z8,sp[8],ss[8],d[16],dq[16];

for (z1=1;z1<=8;z1++)
  { sp[1]=z1;
    for (z2=1;z2<=8;z2++)
     { sp[2]=z2;
       for (z3=1;z3<=8;z3++)
        { sp[3]=z3;
          for (z4=1;z4<=8;z4++)
           { sp[4]=z4;
             for (z5=1;z5<=8;z5++)
               { sp[5]=z5;
                  for (z6=1;z6<=8;z6++)
                   { sp[6]=z6;
                     for (z7=1;z7<=8;z7++)
                       { sp[7]=z7;
                         for (z8=1;z8<=8;z8++)
                          { sp[8]=z8;g++;
                            for (i=1;i<=8;i++) ss[i]=0; /* Test, ob in einer Spalte >1 Dame */
                            for (i=1;i<=8;i++) ss[sp[i]]=1;
                            s=0;for (i=1;i<=8;i++) s+=ss[i];
                            if (s==8)
                            { for (i=1;i<=16;i++) dq[i]=0; /* Test, ob in Quer-Diagonalen >1 Dame */
                              for (i=1;i<=8;i++) dq[i+9-sp[i]]=1;
                              s=0;for (i=1;i<=16;i++) s+=dq[i];
                              if (s==8)
                               { for (i=1;i<=16;i++) d[i]=0; /* Test, ob in Diagonalen >1 Dame */
                                 for (i=1;i<=8;i++) d[i+sp[i]]=1;
                                 s=0;for (i=1;i<=16;i++) s+=d[i];
                                 if (s==8)
                                   { anz++;  /* Lösung gefunden */
                                     printf("\n\n  IaIbIcIdIeIfIgIhI");
                                     for (i=1;i<=8;i++)
                                       {printf("\n%i ",i);
                                        for (j=1;j<=8;j++)
                                            if (sp[i]==j) printf("ID"); else printf("I_");
                                        printf("I");
                                       }}}}}}}}}}}}}}}}
    printf("\n\nAnzahl der Lösungen = %i von insgesamt %i", anz,g);
}
```



Quer-Diagonalen

j= \ i=	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	10
3	4	5	6	7	8	9	10	11
4	5	6	7	8	9	10	11	12
5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14
7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	16

i + j

Diagonalen

j= \ i=	1	2	3	4	5	6	7	8
1	9	8	7	6	5	4	3	2
2	10	9	8	7	6	5	4	3
3	11	10	9	8	7	6	5	4
4	12	11	10	9	8	7	6	5
5	13	12	11	10	9	8	7	6
6	14	13	12	11	10	9	8	7
7	15	14	13	12	11	10	9	8
8	16	15	14	13	12	11	10	9

9 + i - j

```
  IaIbIcIdIeIfIgIhI
1 IDI_I_I_I_I_I_I_I
2 I_I_I_I_IDI_I_I_I
3 I_I_I_I_I_I_I_IDI
4 I_I_I_I_I_IDI_I_I
5 I_I_IDI_I_I_I_I_I
6 I_I_I_I_I_I_IDI_I
7 I_IDI_I_I_I_I_I_I
8 I_I_I_IDI_I_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 IDI_I_I_I_I_I_I_I
2 I_I_I_I_I_I_IDI_I
3 I_I_I_I_IDI_I_I_I
4 I_I_I_I_I_I_I_IDI
5 I_IDI_I_I_I_I_I_I
6 I_I_I_IDI_I_I_I_I
7 I_I_I_I_I_IDI_I_I
8 I_I_IDI_I_I_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_IDI_I_I_I
3 I_I_I_I_I_I_IDI_I
4 I_I_I_IDI_I_I_I_I
5 IDI_I_I_I_I_I_I_I
6 I_I_I_I_I_I_I_IDI
7 I_I_I_I_I_IDI_I_I
8 I_I_IDI_I_I_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_I_I_IDI_I
3 I_I_IDI_I_I_I_I_I
4 I_I_I_I_I_IDI_I_I
5 I_I_I_I_I_I_I_IDI
6 I_I_I_I_IDI_I_I_I
7 IDI_I_I_I_I_I_I_I
8 I_I_I_IDI_I_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 IDI_I_I_I_I_I_I_I
2 I_I_I_I_IDI_I_I_I
3 I_I_I_I_I_IDI_I_I
4 I_I_IDI_I_I_I_I_I
5 I_I_I_I_I_I_IDI_I
6 I_I_I_IDI_I_I_I_I
7 I_IDI_I_I_I_I_I_I
8 I_I_I_I_IDI_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_IDI_I_I_I_I
3 I_I_I_I_I_IDI_I_I
4 I_I_I_I_I_I_I_IDI
5 I_I_IDI_I_I_I_I_I
6 IDI_I_I_I_I_I_I_I
7 I_I_I_I_I_I_IDI_I
8 I_I_I_I_IDI_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_I_IDI_I_I
3 IDI_I_I_I_I_I_I_I
4 I_I_I_I_I_I_IDI_I
5 I_I_I_IDI_I_I_I_I
6 I_I_I_I_I_I_I_IDI
7 I_I_IDI_I_I_I_I_I
8 I_I_I_I_IDI_I_I_I
```

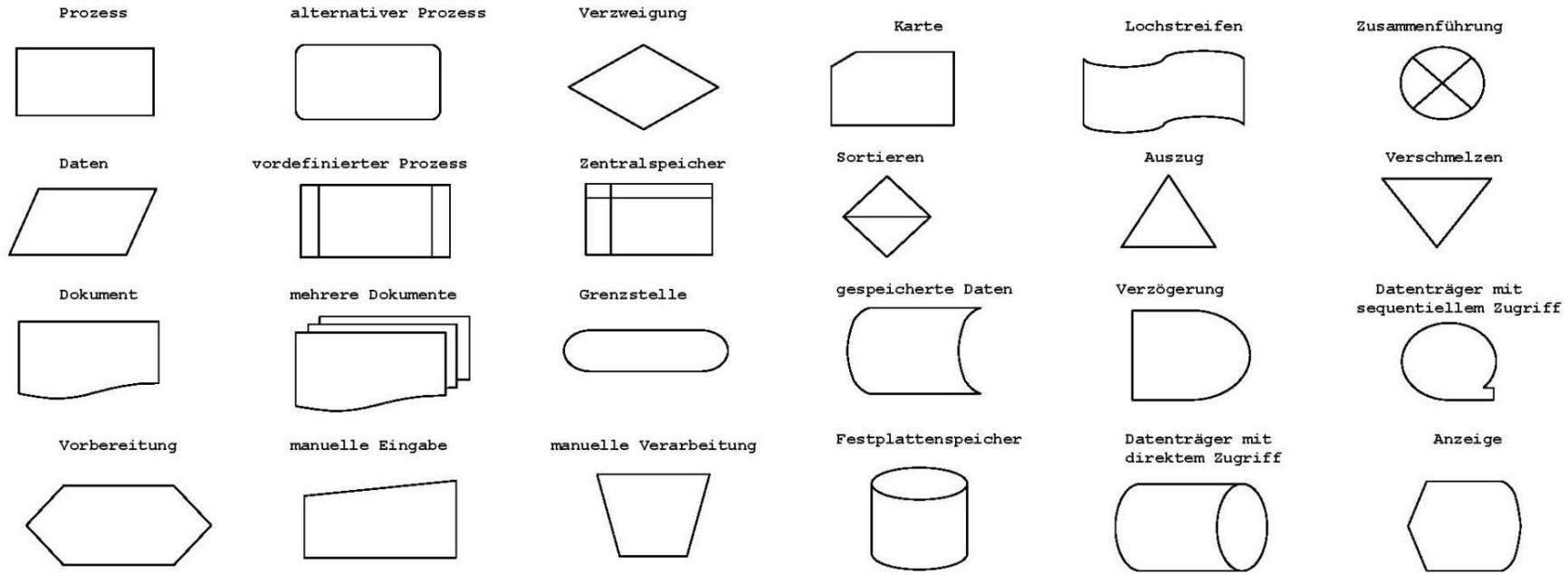
```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_I_I_IDI_I
3 I_I_I_IDI_I_I_I_I
4 I_I_I_I_I_IDI_I_I
5 I_I_I_I_I_I_I_IDI
6 I_I_I_IDI_I_I_I_I
7 I_I_I_I_I_IDI_I_I
8 I_I_IDI_I_I_I_I_I
```

```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_IDI_I_I_I
3 I_I_I_I_I_I_IDI_I
4 I_I_IDI_I_I_I_I_I
5 IDI_I_I_I_I_I_I_I
6 I_I_I_IDI_I_I_I_I
7 I_I_I_I_I_I_IDI_I
8 I_I_I_IDI_I_I_I_I
```

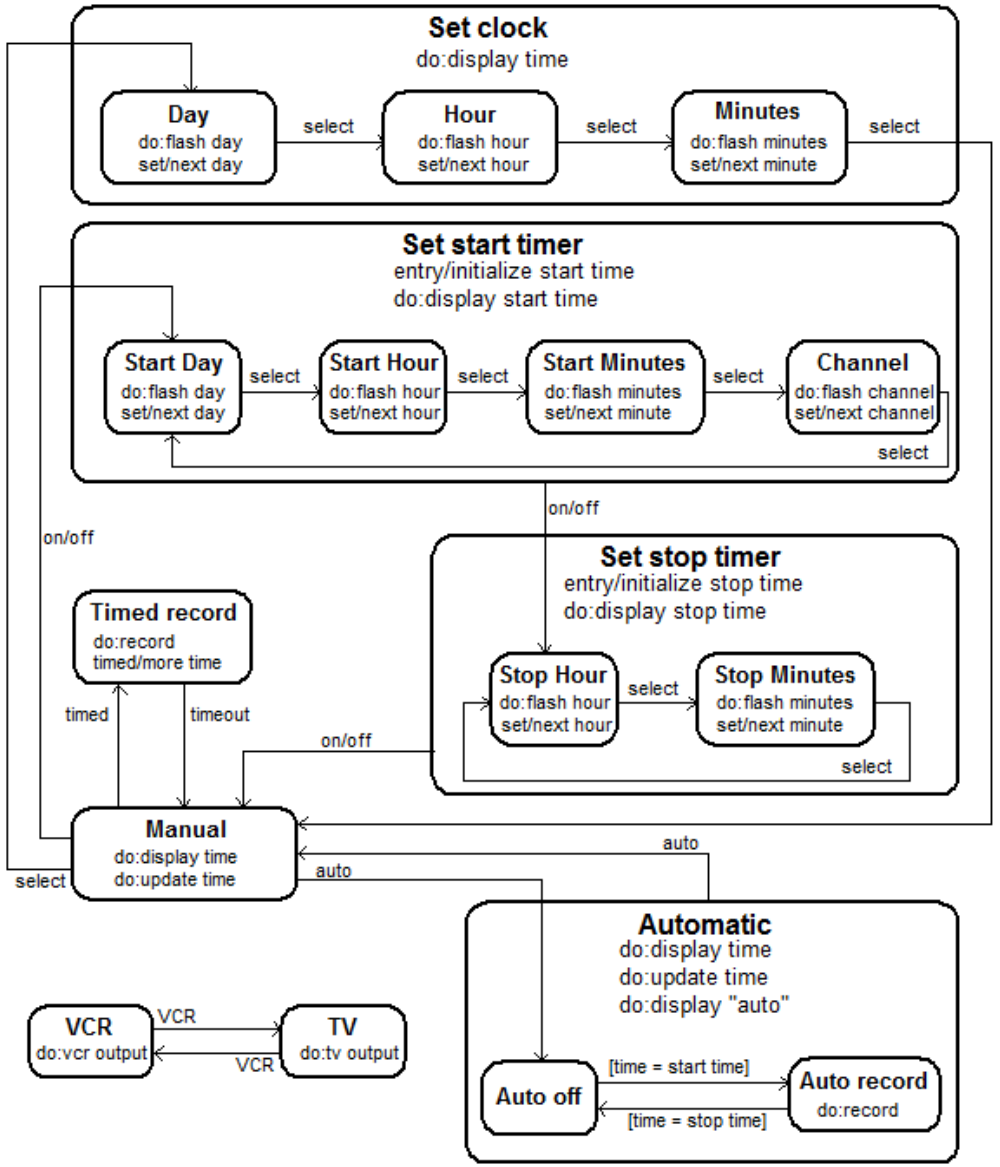
```
  IaIbIcIdIeIfIgIhI
1 I_IDI_I_I_I_I_I_I
2 I_I_I_I_I_I_I_IDI
3 I_I_I_I_I_IDI_I_I
4 IDI_I_I_I_I_I_I_I
5 I_I_IDI_I_I_I_I_I
6 I_I_I_I_I_I_I_IDI
7 I_I_I_I_I_IDI_I_I
8 I_I_I_IDI_I_I_I_I
```

Datenflussdiagramme (DIN 66001 von 1983)

ISO/DIS 5807 1979 ISO 2636 ISO 1028 1973

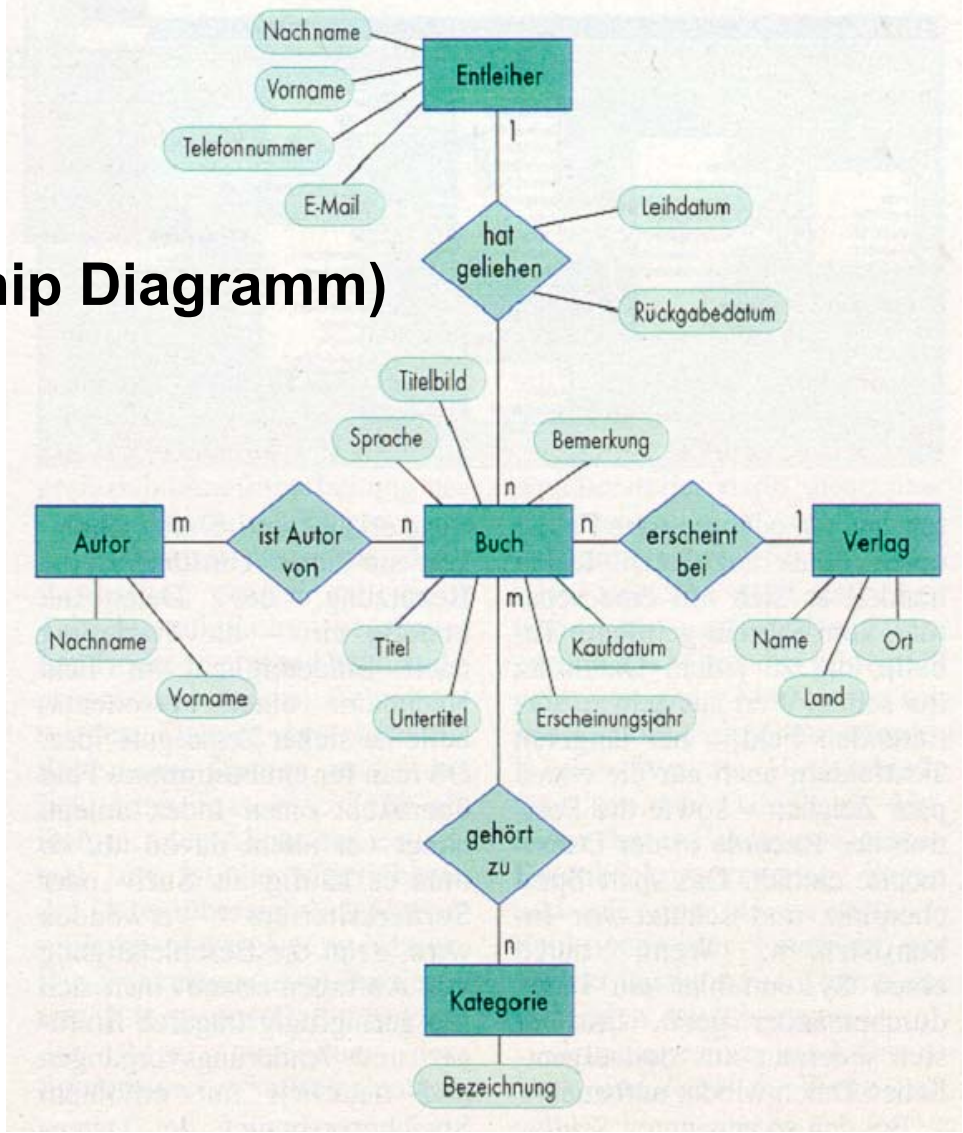


Zustands-diagramm

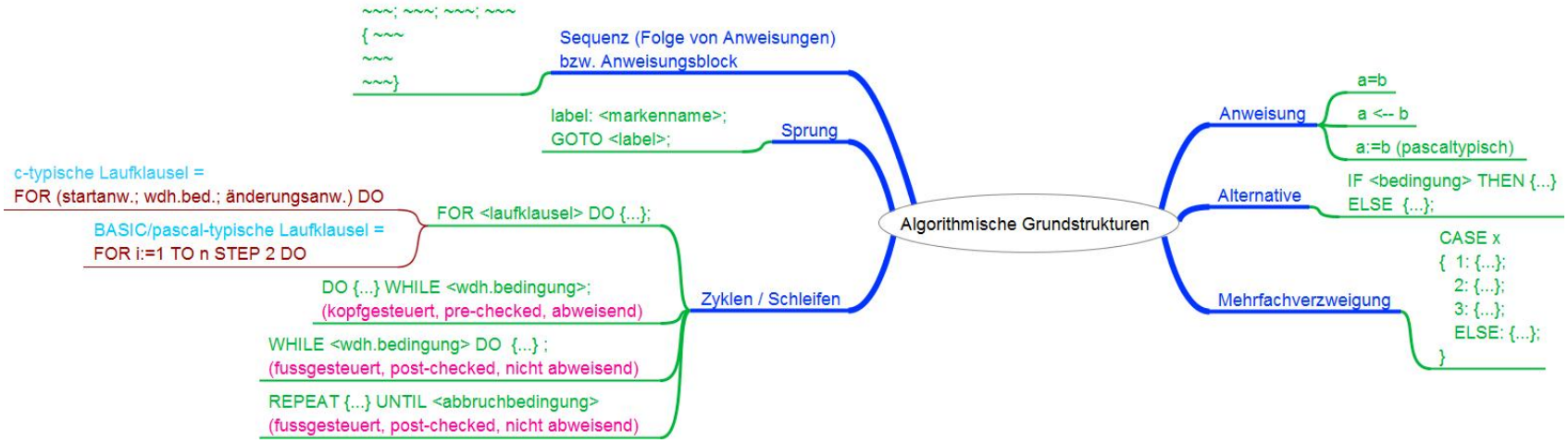


ER-Diagramm

(Entity Relationsship Diagramm)



algorithmische Grundstrukturen



Pseudocode (Algorithmische Grundstrukturen)

Ein- und Ausgabe:	**Eingabe:** a, b, c **Ausgabe:** "TextText", a	**IN:** a, b, c **OUT:** „TextText“, a
Typedefinition	Textfeld = **Feld Char** [1..30]; Matrix = **Feld Real** [1..2,1..10];	
Daten	Name, Vorname: Textfeld; Tabelle: Matrix;	

Anweisung = einfache Programmaktion ;
Sequenz = Folge von Anweisungen, logischen Elementarstrukturen oder komplexen Programmstrukturen
{ anweisung$_1$ anweisung$_2$ anweisung$_3$ **}** (Anweisungsblock)
oder z.B.
Setze a:=1; a:=b; c:=sqrt(sqr(a)-sqr(b));

Wertzuweisung einer Variablen:
a:=1 im Sinne von a ← 1 (Variable a bekommt den Wert 1 zugewiesen)
a:=b im Sinne von a ← b (Variable a bekommt den Wert von b zugewiesen,
d.h. Wert wird dupliziert und evtl. in den Zieltyp Konvertiert)

Alternative ohne sonst-Zweig **Falls** bedingung **dann {** anweisung$_1$ anweisung$_2$ anweisung$_3$ **}**	**IF** bedingung **THEN {**anweisung$_1$;anweisung$_2$;anweisung$_3$**}**
Alternative mit sonst-Zweig **Falls** bedingung **dann {** anweisung$_1$ anweisung$_2$ anweisung$_3$ **}** **sonst {** anweisung$_4$ anweisung$_5$ **}**	**IF** bedingung **THEN {**anweisung$_1$;anweisung$_2$;anweisung$_3$**}** **ELSE {** anweisung$_4$ anweisung$_5$ **}**

Mehrfachverzweigung:

```
Falls selektor
        wert1: anweisungsblock1;
        wert2: anweisungsblock2;
        wert3: anweisungsblock3;
        .....
sonst anweisungsblock_n;
```

```
CASE selektor
        value1: anweisungsblock1;
        value2: anweisungsblock2;
        value3: anweisungsblock3;
        ……..
ELSE anweisungsblock_n;
```

Zyklus:

```
Für i:= 1 solange i<n  mit i:=i+1
führe aus anweisungsblock;
```

```
FOR (i=1;i<n;i++) DO anweisungsblock;
```

```
Solange bedingung führe aus
  anweisungsblock
```

```
WHILE bedingung DO anweisungsblock;
```

```
Wiederhole
  anweisungsblock
solange bedingung;
```

```
DO bedingung WHILE anweisungsblock;
```

Bsp:

```
Für i:=1 solange i≤2 mit i:=i+1 führe aus
        Für j:=1 solange j≤10 mit j:=j+1 führe aus
                Tabelle[i,j]:= 0.0;
```

```
FOR (i=1;i≤2;i++) DO
    FOR (j=1;j≤10;j++) DO
        Tabelle[i,j]:=0.0;
```

Blöcke/Strukturblock:

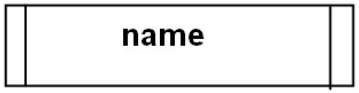
```
Block name (ein: a,b ∈ REAL; aus: T ∈ Tabelle)
  // Kommentar
Daten   x ∈ Integer;
        x ∈ Real;
{

}
```

Struktogramme nach Nassi-Shneidermann (DIN 66261)

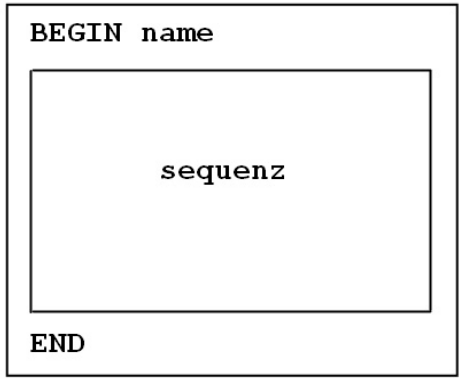
Elementarblock

Physische Block-Einfügung

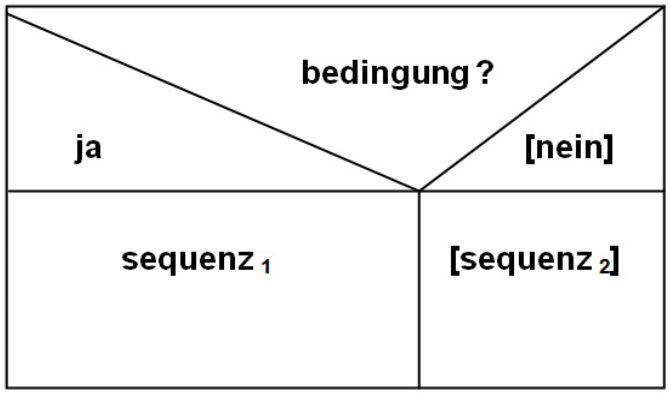
Unterprogrammaufruf



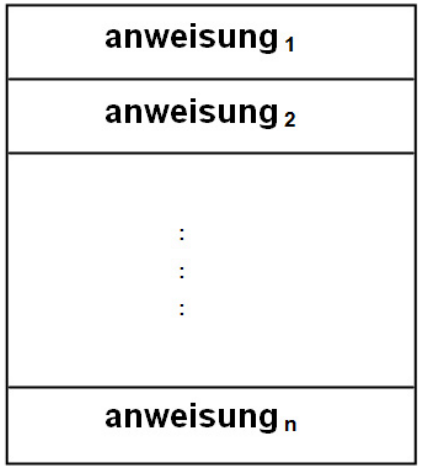
Strukturbblock



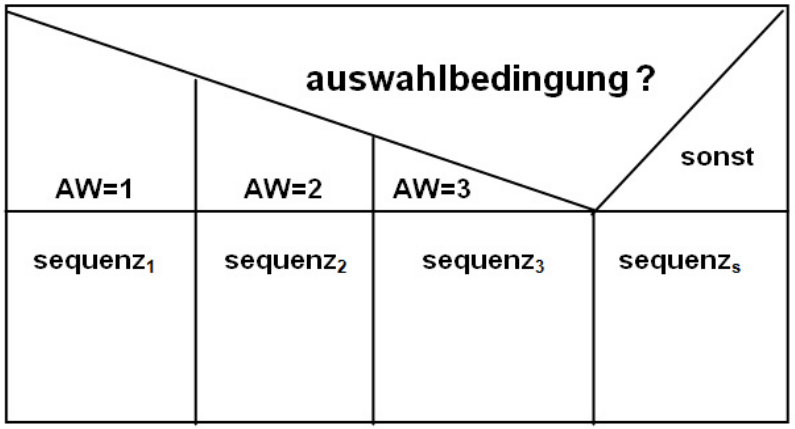
Alternative



Sequenz = Anweisungsblock

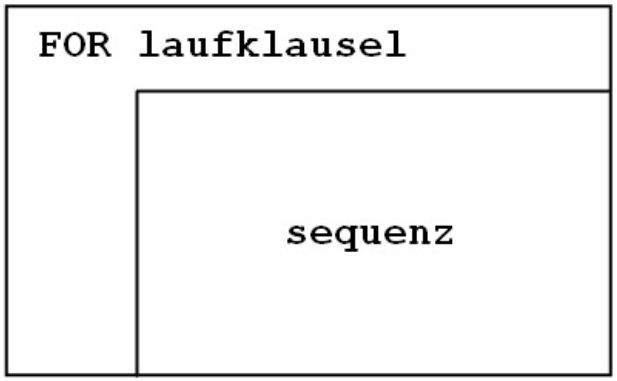


Mehrfachverzweigung/Selektion

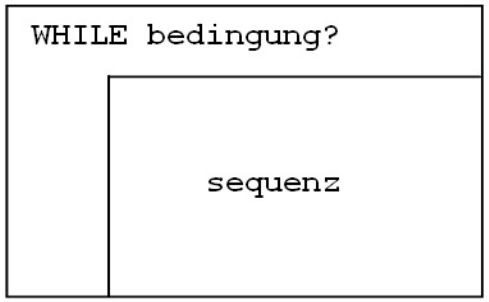


Zyklus

laufklausel:= (LV=A bis E mit S)
oder (LV=A;Wdh.bed.;Veränderung)



abweisender Zyklus
(Solange die Bedingung erfüllt ist wiederhole)



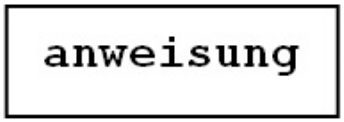
nichtabweisender Zyklus
(Wiederhole bis Endebedingung erreicht ist)

nichtabweisender Zyklus
(Wiederhole, solange die Bedingung erfüllt ist)

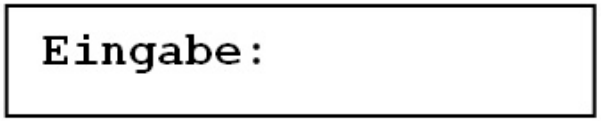
PAP – Programmablaufplan/ Datenflußdiagramme (Sinnbilder nach DIN 66001)

Grenzstellen nach aussen:

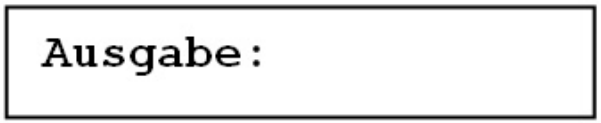
Anweisung/Aktion:



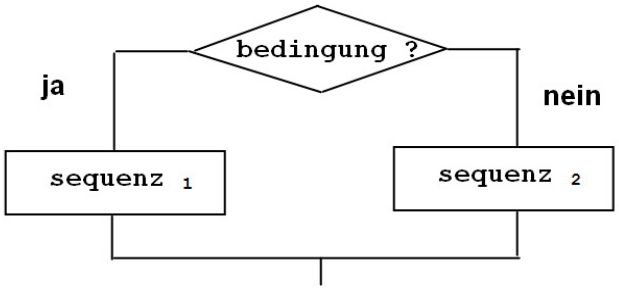
Dateneingabe:



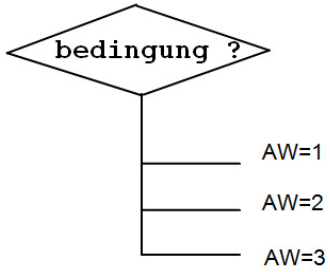
Datenausgabe:



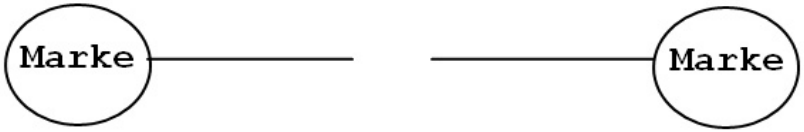
Alternative:



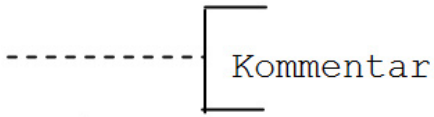
Mehrfachauswahl/Selektion:



Programmverbindung/
Programmverlauf:



Kommentar:



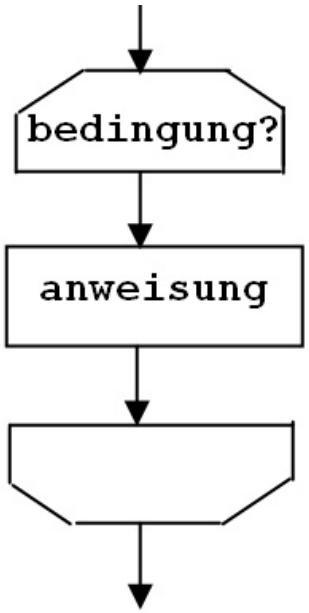
Sprung mit Rückkehr:

Sprung ohne Rückkehr:

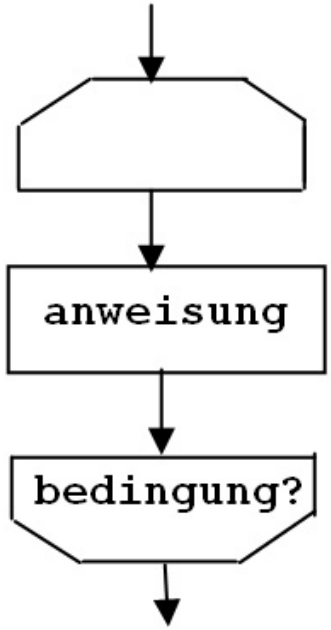
Unterbrechung:

Zyklen

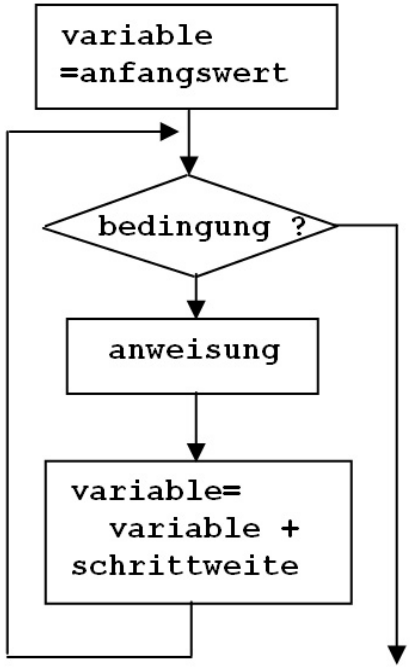
abweisender Zyklus ("pre-checking")



nicht-abweisender Zyklus ("post-checking")



FOR-Zyklus



Basisalgorithmen in Pseudocode

Zählen:

Setze z:=0;
Für i:=1 **solange** i ≤ n **mit** i:=i+1 **führe aus**
Falls a_i der Bedingung genügt **dann setze** z:=z+1;
Ausgabe: "Anzahl der Elemente, die der Bedingung genügen=", z

```
z=0;
FOR (i=1;i≤n;i++) DO
 {IF (ai<0) THEN z=z+1;}
out: .....
```

Summieren

Setze s:=0;
Für i:=1 **solange** i ≤ n **mit** i:=i+1 **führe aus**
Setze s:=s+a_i;
Ausgabe: "Summe der Elemente=", s

```
S=0;
FOR (i=1; i≤n; i++) DO S+=ai;
OUT: "Summe der Elemente =", S
```

Vertauschen **Setze** aa:=a_i; a_i:=a_j; a_j:=aa;

Verdichten **Für** i:=k **solange** i ≤ n-1 **mit** i:=i+1 **führe aus setze** a_i:=a_{i+1};
Setze n:=n-1;

. . .

Einfügen **Für** i:=n **solange** i ≥ k **mit** i:=i-1 **führe aus setze** a_{i+1}:= a_i; //rückwärts Lücke schaffen
Setze a_K:=aa; n:=n+1;

Aufsplitten

Setze k1:=0; k2:=0;
Für i:=1 **solange** i ≤ n **mit** i:=i+1 **führe aus**
Falls a_i der Bedingung genügt **dann**
Setze k1:=k1+1; b_{k1}:=a_i;
Sonst k2:=k2+1; c_{k2}:=a_i;

```
k1:=0; k2:=0;
FOR (i=1;i<=n;i++) DO
 IF (ai der Bedingung genügt) THEN
        k1:=k1+1; bk1:=ai;
 ELSE k2:=k2+1; ck2:=ai;
```

Maximum-suche

Setze MAX:=a_1; POS:=1;
Für i:=2 **solange** i ≤ n **mit** i:=i+1 **führe aus**
Falls a_i > MAX **dann setze** MAX=a_i;POS=i;

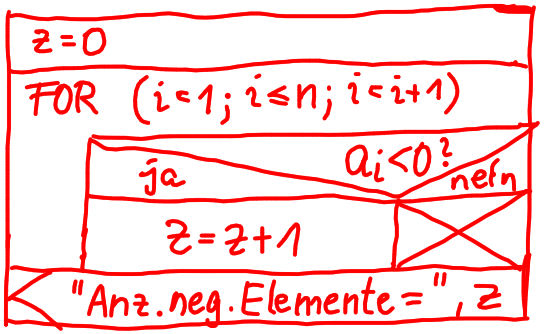
Setze MAX:=-1E30; POS:=0;
Für i:=1 **solange** i ≤ n **mit** i:=i+1 **führe aus**
Falls a_i > MAX **dann setze** MAX=a_i;POS=i;

Basisalgorithmen in Struktogrammform

Vertauschen

$aa := a_i$	**K: rette a_i**
$a_i := a_j$	**K: Vertausche**
$a_j := aa$	**K: Zurückholen von aa**

Zählen



Aufsplitten . . .

Produkt . . .

Einfügen . . .

Summieren

```
s:=0
FOR (i:=1; i ≤ n;i:=i+1)
    s:=s+ai
A: "Summe der Elemente =", s
```

Verdichten

```
FOR (i:=k; i ≤ n-1;i:=i+1)
    ai:=ai+1
```

Maximumsuche

```
MAX = a1 ; POS = 1;
FOR (i:=2; i ≤ n;i:=i+1)
    ai > MAX ?
    ja                nein
    MAX = ai
    POS = i
```