1. Grundlegende Konzepte der Informatik

Inhalt

- Algorithmen
- Darstellung von Algorithmen mit Programmablaufplänen
- Beispiele für Algorithmen
- Aussagenlogik
- Zahlensysteme
- Kodierung

Algorithmen

Einführende Beispiele

Täglich werden Verarbeitungsvorschriften ausgeführt, die einen Weg beschreiben, wie ein gewünschtes Ergebnis erzielt werden kann.
Solche Vorschriften sind:

- **Bedienungsanleitungen**

  zum Beispiel zur Bedienung eines Parkscheinautomaten
- **Montageanleitungen**

  Aufbau von Möbelstücken, Beispiele im Modellbaukasten
- **Verhaltensvorschriften**

  Verhalten bei Unfällen, Feueralarm, usw.
- **Rezepte**

  Mischen von Farben, Kochrezepte, usw.
- **Rechenvorschriften**

  Lösen eines Gleichungssystems, Berechnung eines elektrischen Stromkreises, Bestimmung von Flächeninhalten, usw.

Algorithmen

Definition:

Ein Algorithmus ist eine Vorschrift zur Lösung einer Klasse von Problemen. Er besteht aus einer endlichen Folge von Schritten, mit der aus bekannten Eingangsdaten neue Ausgangsdaten eindeutig berechnet werden können.

Eigenschaften:

- Die Definition zeigt den engen Zusammenhang mit **Funktionen** (Eingangsdaten → Ausgangsdaten).
- Die Definition bindet den Algorithmus an die Lösung einer Klasse von Problemen und nicht an eine einzelne Aufgabe.

Problem:

- Die Definition eines Algorithmus enthält keine Forderungen an die praktische Ausführbarkeit des Algorithmus auf einer realen Maschine (Rechner).

Algorithmen als mathematische Funktionen (1)

Ein Algorithmus kann als Funktion (mathematische Definition) aufgefasst werden.

$y=f(x)$ oder $f: D \rightarrow Z,\ x \rightarrow y$

(D: Definitionsmenge, Z: Zielmenge)

In der Mathematik ist eine **Funktion** oder **Abbildung** eine Beziehung zwischen zwei Mengen, die jedem Element der einen Menge (Funktionsargument, unabhängige Variable, x-Wert) genau ein Element der anderen Menge (Funktionswert, abhängige Variable, y-Wert) zuordnet.

(Quelle: de.wikipedia.org)

Interpretation:

- Elemente aus Mengen: Eingabe und Ausgabe
- Die Elemente können auch Mengen, Vektoren o.ä. sein

Algorithmen als mathematische Funktionen (2)

Somit wird einer Funktion ein Algorithmus zugeordnet.

Es gibt aber Funktionen, denen kein Algorithmus zugeordnet werden kann. Diese Funktionen sind damit nicht berechenbar.

Definition:

Eine Funktion f heißt berechenbar, wenn es einen Algorithmus gibt, der für jedes Argument x den Funktionswert f(x) berechnet.

Beispiele folgen …

Beispiele berechenbarer und nicht berechenbarer Funktionen (1)

a) Berechne alle Primzahlen im Intervall [1,100] !
Dieses Beispiel stellt eine berechenbare Funktion dar, der Algorithmus ist relativ einfach.

b) Berechne die Funktionswerte der Funktion
$f(x+1) = 2^{f(x)}$ mit $f(0)=0$ für $6<=x<=12$!
Dies ist auch eine berechenbare Funktion, die sich aber auf keiner realen Rechenmaschine abarbeiten lässt, da f(6) bereits eine Zahl mit 19000 Dezimalstellen ist.

c) „Berechne“ alle reellen Zahlen im Intervall [0,1]
Dieses Beispiel ist eine nicht-berechenbare Funktion dar.
Es kann bewiesen werden, dass solch ein Algorithmus nicht existieren kann.

Beispiele berechenbarer und nicht berechenbarer Funktionen (2)

Schlussfolgerungen:

- Einige Funktionen sind prinzipiell nicht berechenbar.
- Manche Funktionen sind unter realistischem Ressourcen- und Zeitaufwand nicht auf einem Rechner umsetzbar, obwohl sie prinzipiell berechenbar sind.
- Es ist zweckmäßig, an Algorithmen solche Forderungen zu stellen, dass sie auf realen Maschinen praktisch ausgeführt werden können.
  Im folgenden werden diese wichtigen Eigenschaften aufgeführt.

Charakteristische Eigenschaften von Algorithmen

Endlichkeit

Ein Algorithmus muss aus einer endlichen Anzahl von Lösungsschritten bestehen und nach Abarbeitung dieser endlich vielen Schritte nach einer endlichen Zeit das Ende erreichen.

Eindeutigkeit

Die einzelnen Schritte eines Algorithmus und ihre Aufeinanderfolge müssen eindeutig beschrieben sein.

Allgemeinheit

Ein Algorithmus darf nicht nur die Lösung einer speziellen Aufgabe (z.B. Lösung der Gleichung $x^2 + 2x + 1=0$) , sondern muss die Lösung einer Klasse von Problemen (z.B. die Lösung aller quadratischen Gleichungen $ax^2 +bx +c =0$) beschreiben.

Determiniertheit

Die mehrmalige Anwendung des Algorithmus mit denselben Eingangsdaten muss immer wieder dieselben Ausgangsdaten liefern.

Charakteristische Eigenschaften von Algorithmen

Effizienz

Ein Algorithmus muss möglichst wenig Ressourcen einer Maschine, d.h. möglichst wenig Rechenzeit und möglichst wenig Speicher in Anspruch nehmen.

In der Informatik wurden Effizienzmaße für Algorithmen, die sogenannte Zeit- und Speicherkomplexität entwickelt, mit der Algorithmen bewertet werden können. Dies ist praktisch sehr wichtig, da i.a. zur Berechnung einer Funktion mehrere Algorithmen angegeben werden können.

Die Darstellung von Algorithmen

Aus den Einführungsbeispielen und Übungsaufgaben ist erkennbar, dass zur Darstellung von Algorithmen Grundelemente notwendig sind. Neben der Notation einzelner **elementarer Aktionen/Anweisungen**, wie z.B.

- Gebe die Werte der Koeffizienten a,b,c ein

sind auch **logische Elementarstrukturen** notwendig, die die **zeitliche Ablauffolge** darstellen, wie z.B. die Folge (**Sequenz**) von Anweisungen, wie z.B.

1. Gebe die Werte der Koeffizienten a,b,c ein
2. Berechne d = b*b – 4*a*c

aber auch die Auswahl (**Selektion**)

3. Falls d<0 setze fort mit Schritt 7

formale Darstellung von Algorithmen

Bei der Darstellung von Algorithmen liegt der Schwerpunkt vor allem auf der Darstellung der zeitlichen Ablauffolge, dem sogenannten ***Steuerfluss*** und nicht auf der Darstellung der einzelnen elementaren Anweisung an sich.

Theorem von Böhm und Jacopini

Mit nur drei logischen Elementarstrukturen

- **Sequenz**
- **Selektion**
- **Zyklus**

lässt sich jedes algorithmierbare Problem darstellen.

formale Darstellung von Algorithmen

Es wurden zwei verschiedene Darstellungsformen entwickelt, die hauptsächlich in Gebrauch sind und auf grafische Symbole zurückgreifen:

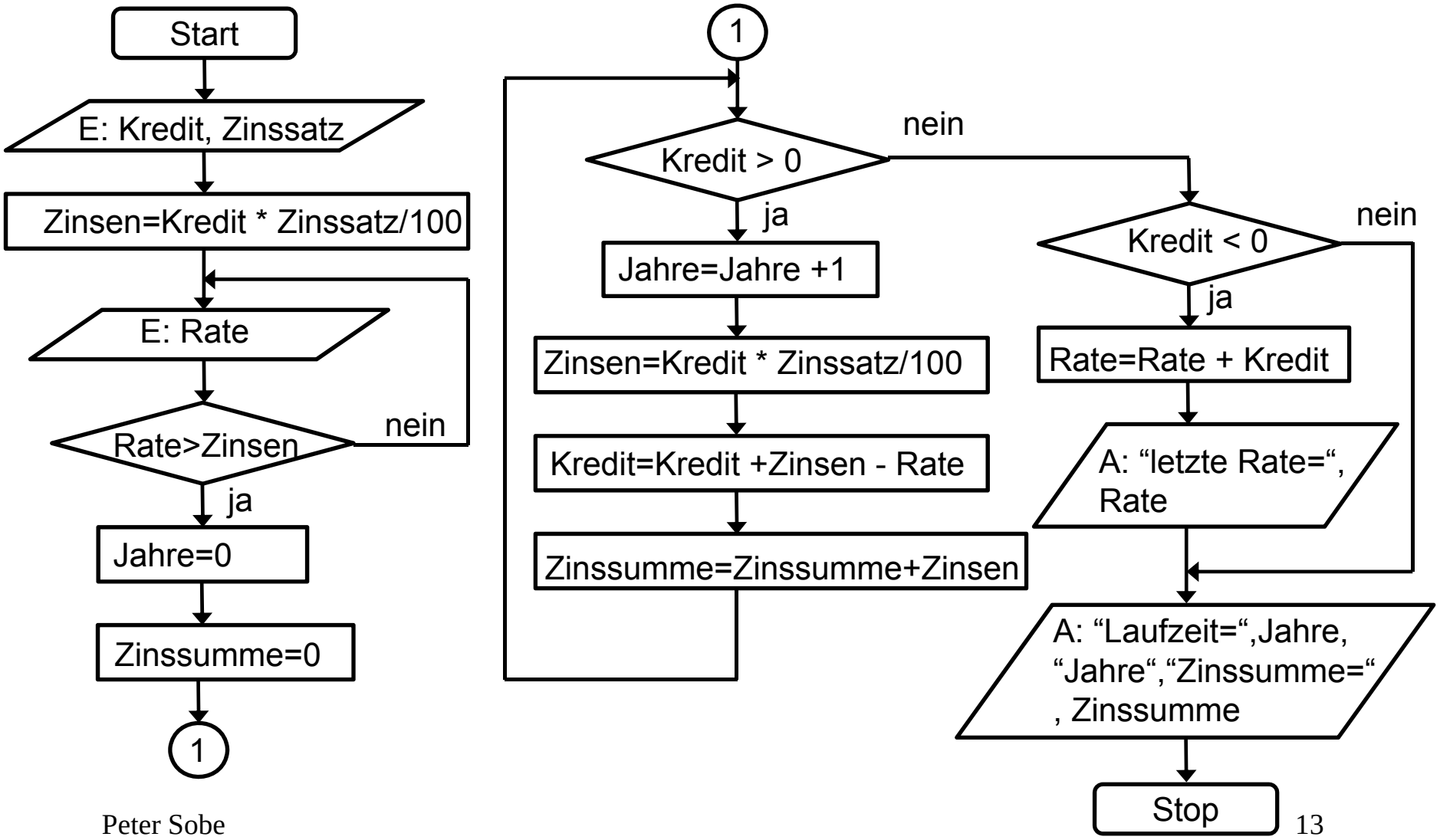
- **Programmablaufplan (abk. PAP)**
  nach DIN 66001 (vgl. Bild 1)

- **Struktogramm** nach Nassi und Shneiderman
  nach DIN 66261 (vgl. Bild 2)

Beide repräsentieren eine sogenannte ***Syntax***, die die Ausdrucksformen der Beschreibung formal spezifiziert.

Die Bedeutung der Darstellung wird als ***Semanti***k bezeichnet.

Ein Algorithmus als Programmablaufplan

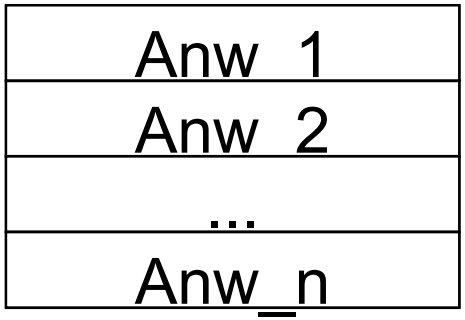


Algorithmus als Struktogramm

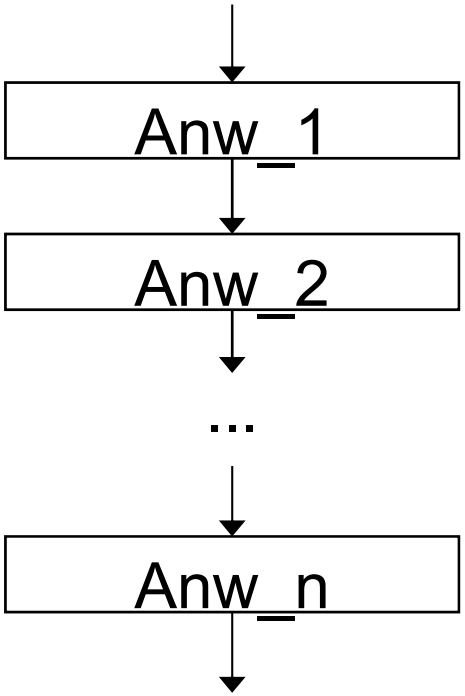
Struktogramm

Sequenz

Struktogramm



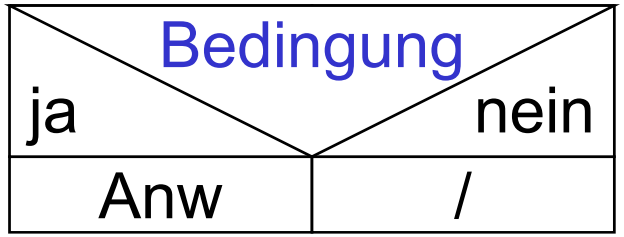
Programmablaufplan



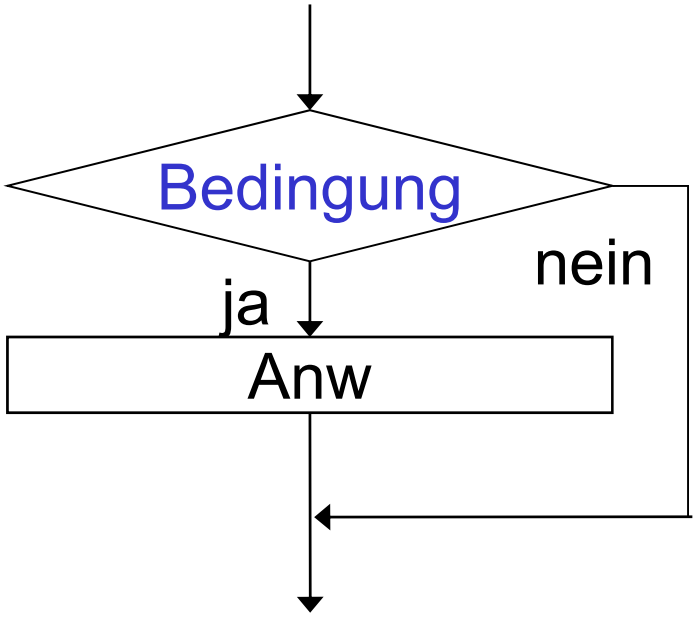
Das Wesentliche einer Sequenz besteht darin, dass sie nach außen wiederum nur eine einzige Aktion/Anweisung repräsentiert, obwohl sie intern aus einer Folge von Anweisungen (Anw) besteht.

einseitige Selektion

Struktogramm



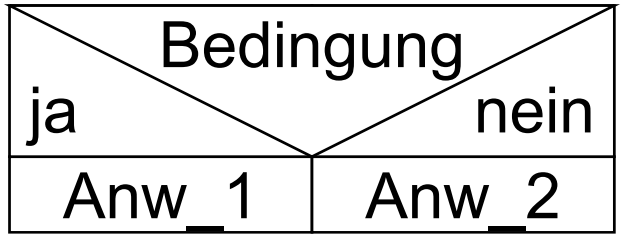
Programmablaufplan



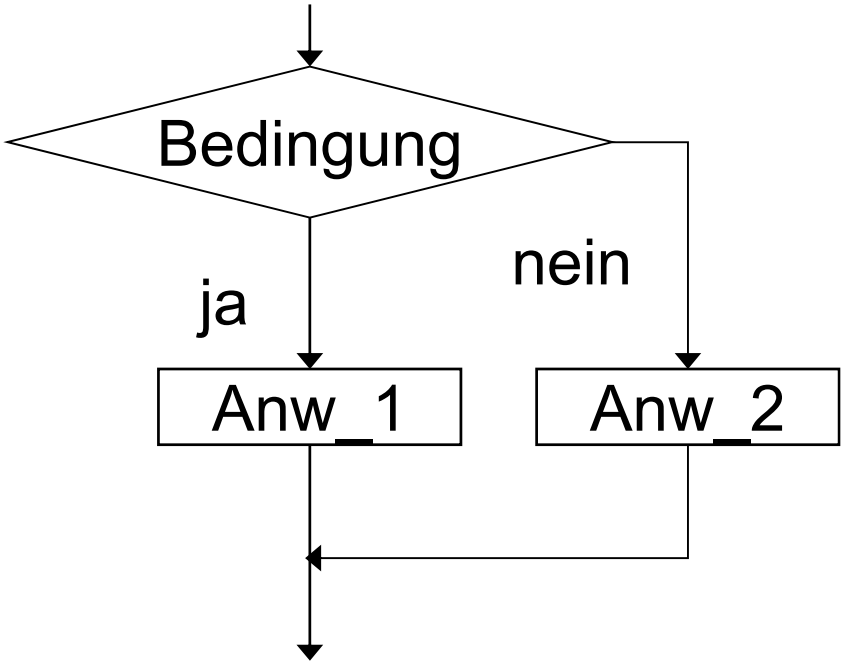
Die einseitige Selektion gestattet die Ausführung einer Anweisung *Anw* (welche wiederum eine Sequenz sein kann), wenn die angegebene Bedingung wahr (ja) ist.

zweiseitige Selektion

Struktogramm



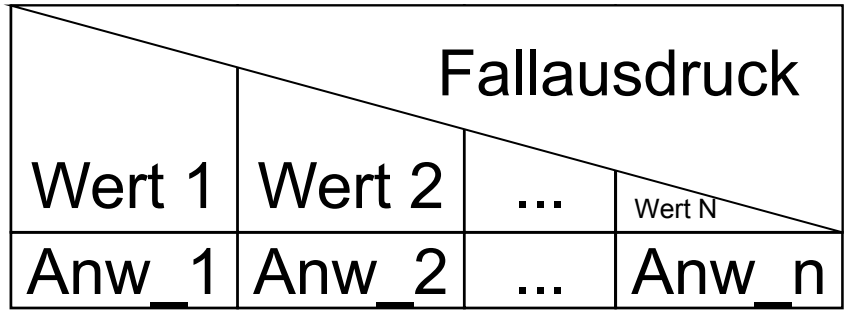
Programmablaufplan



Die zweiseitige Selektion gestattet die Auswahl einer Anweisung (welche wiederum eine Sequenz sein kann) aus zwei angegebenen Sequenzen *Anw_1* bzw. *Anw_2*.

Wenn die angegebene Bedingung wahr (ja) ist, wird *Anw_1* ausgeführt, sonst *Anw_2*.

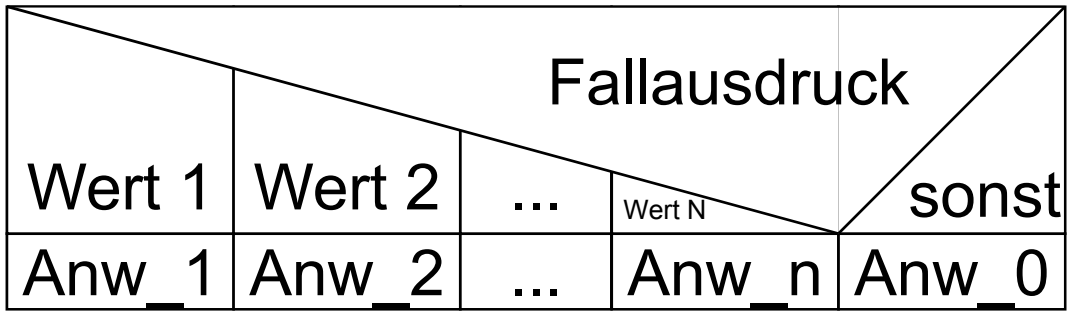
mehrfache Selektion ohne Sonst- Zweig



Dieses Konstrukt besitzt in der PAP-Darstellung keine direkte Entsprechung !

Der Fallausdruck kann einfache Werte repräsentieren, z.B. ganzzahlige oder dezimale Werte, ein Zeichen, eine Zeichenkette. Für alle zu behandelnden Fälle wird eine entsprechende Konstante als *Wert 1*, *Wert 2* usw. angegeben. Entsprechend dem zutreffenden Wert wird eine angegebene Anweisung ausgeführt, also *Anw_1* oder *Anw_2* usw.

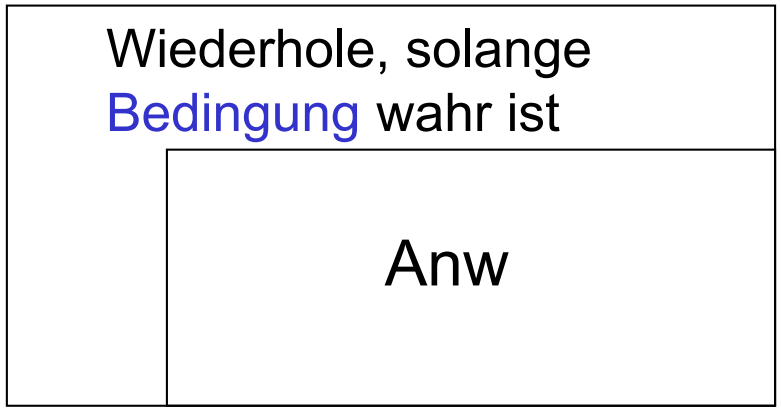
mehrfache Selektion mit Sonst-Zweig



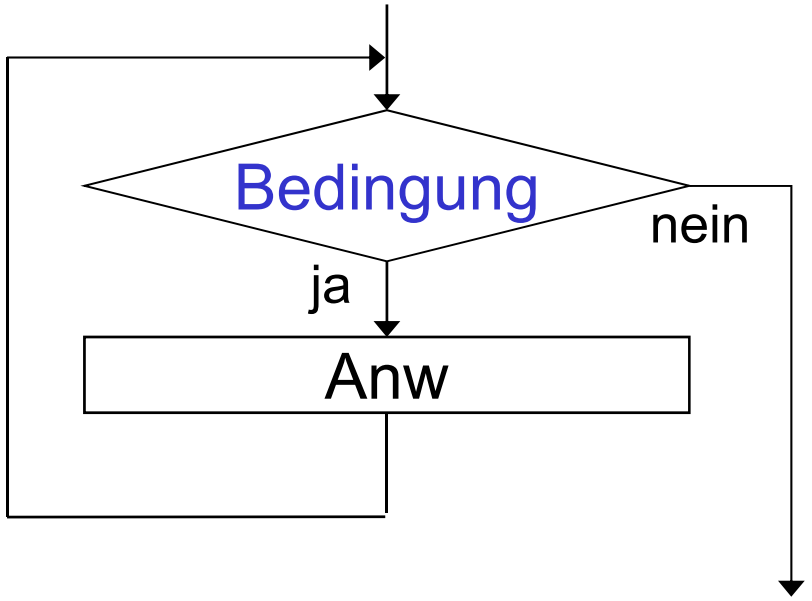
Die Semantik entspricht im Prinzip der mehrfachen Selektion ohne Sonst-Zweig mit der Erweiterung, dass wenn der Fallausdruck keinen der angegebenen Werte *Wert 1*, *Wert 2*, ... repräsentiert, wird dann die Anweisung *Anw_0* ausgeführt.

Zyklus abweisend

Struktogramm



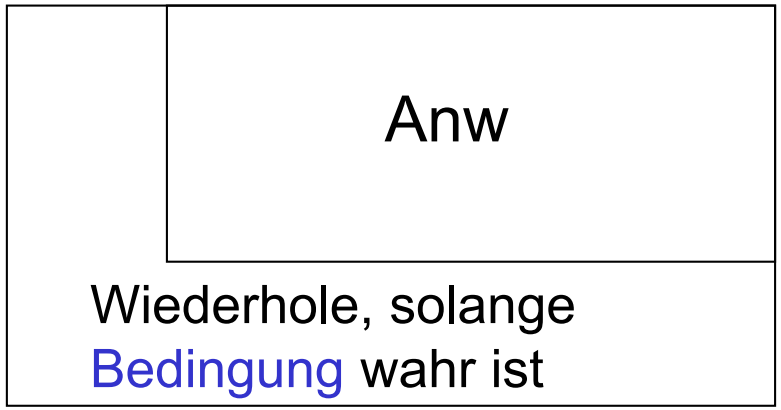
Programmablaufplan



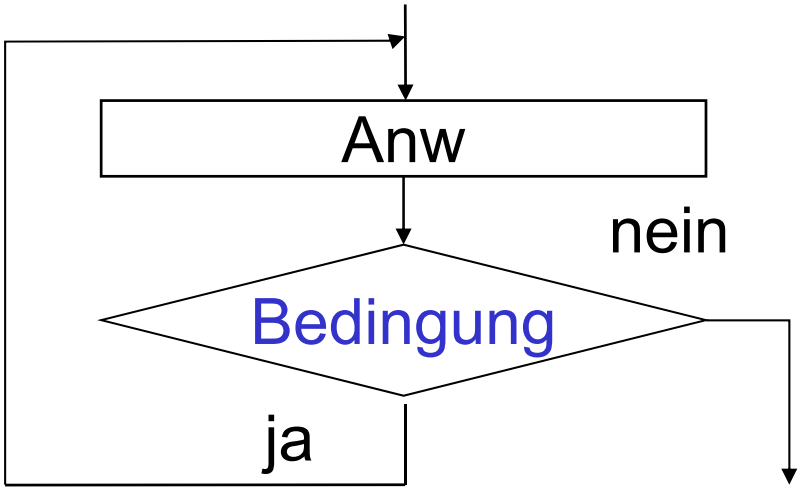
Die Ausführung einer Anweisung *Anw* (welche wiederum eine Sequenz sein kann), wird solange wiederholt ausgeführt, solange die angegebene Bedingung wahr (ja) ist. Abweisend bedeutet, dass wenn die Bedingung bei Eintritt bereits falsch ist, wird Anw überhaupt nicht ausgeführt.

Zyklus nicht abweisend

Struktogramm



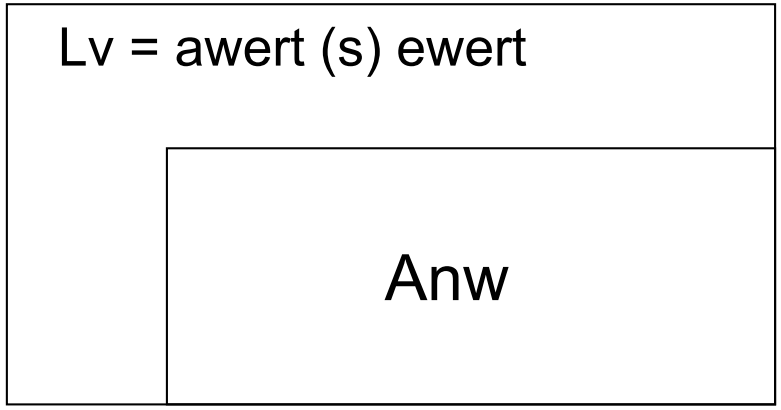
Programmablaufplan



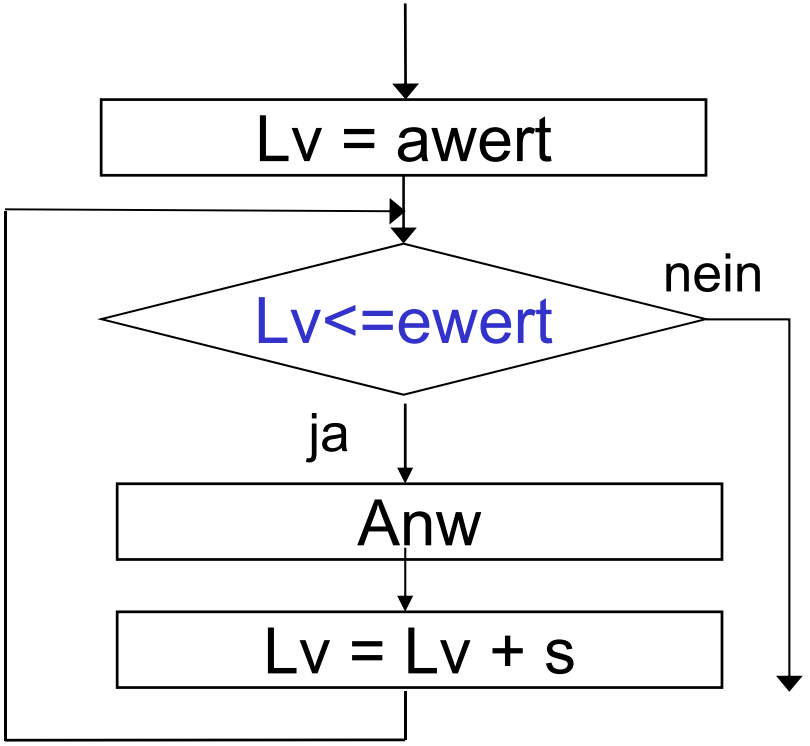
Die Ausführung des Zyklus beginnt mit der Ausführung der Anweisung Anw (welche wiederum eine Sequenz sein kann), ohne Prüfung der Bedingung (deshalb „**nicht** abweisend“). Erst nach der Ausführung von *Anw* wird die angegebene Bedingung geprüft. Ist die Bedingung wahr, wird die Anweisung Anw wieder ausgeführt. Das wird wiederholt, solange die angegebene Bedingung wahr (ja) ist.

Zähl-Zyklus

Struktogramm



Programmablaufplan



Lv die ist sogenannte Laufvariable

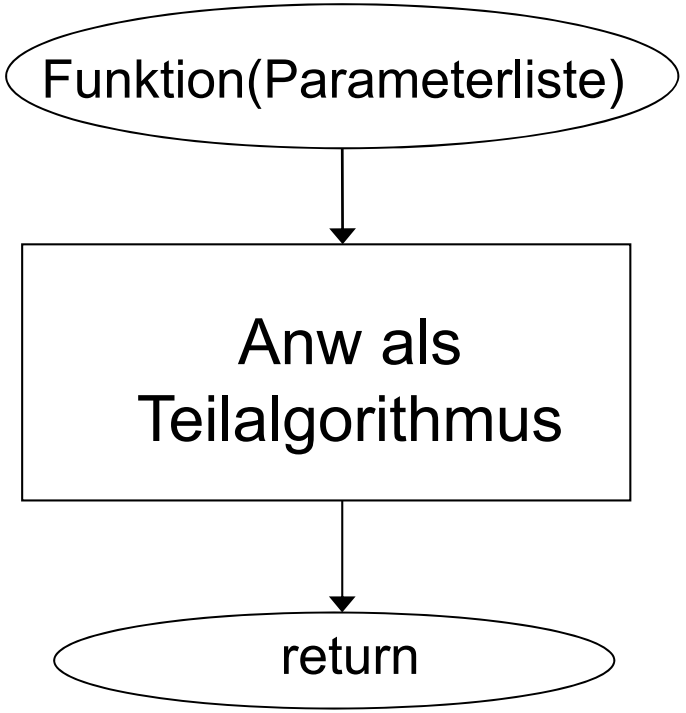
- **awert** ist der Anfangswert von Lv
- **s** ist die Schrittweite
- **ewert** ist der Endwert von Lv

Die Anweisung *Anw* erfährt eine bestimmte Anzahl von Wiederholungen, die sich aus den Größen awert, s und ewert ergibt.

Prozedur-Deklaration

Struktogramm

Programmablaufplan

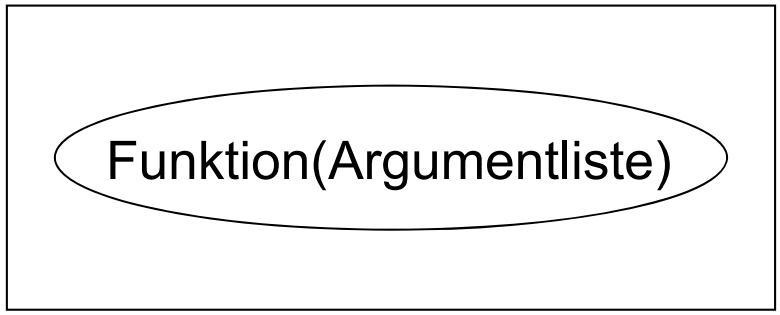


Funktion ist der Name eines Teilalgorithmus. Alle Größen, die der Algorithmus aus der Umgebung zur Abarbeitung benötigt, müssen in der Parameterliste definiert werden.

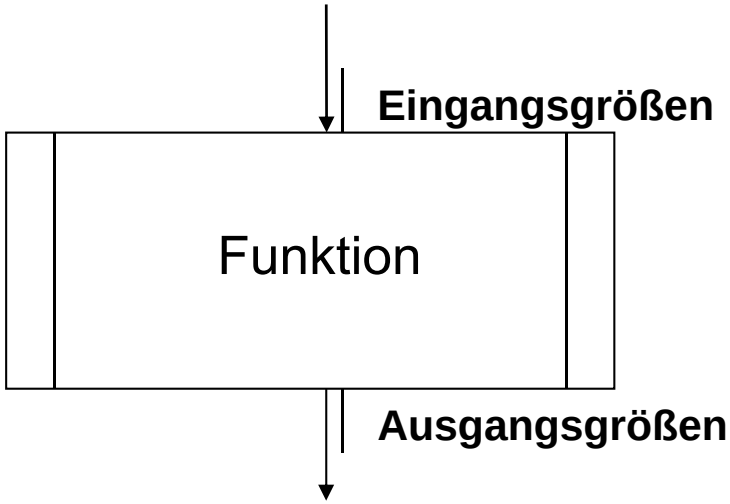
In der Parameterliste müssen Eingangs- und Ausgangsgrößen unterschieden werden.

Prozedur-Aufruf

Struktogramm



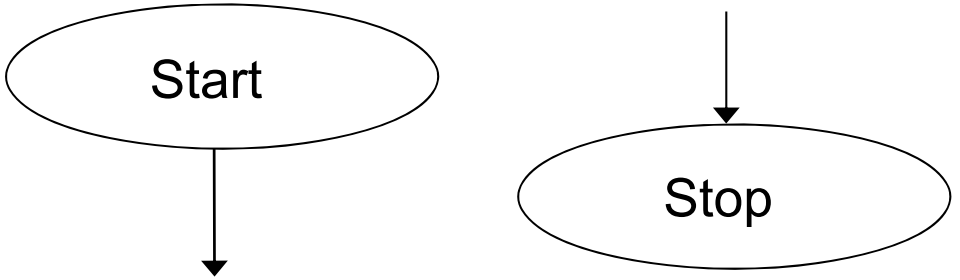
Programmablaufplan



Die Argumente sind die in einer Umgebung gültigen Objekte, die für Parameter beim Aufruf eingesetzt werden. Dabei müssen in der Argumentliste Eingangs- und Ausgangsgrößen unterschieden werden.

Hilfsgrößen bei Programmablaufplänen

Anfang / Ende
eines Hauptprogrammes



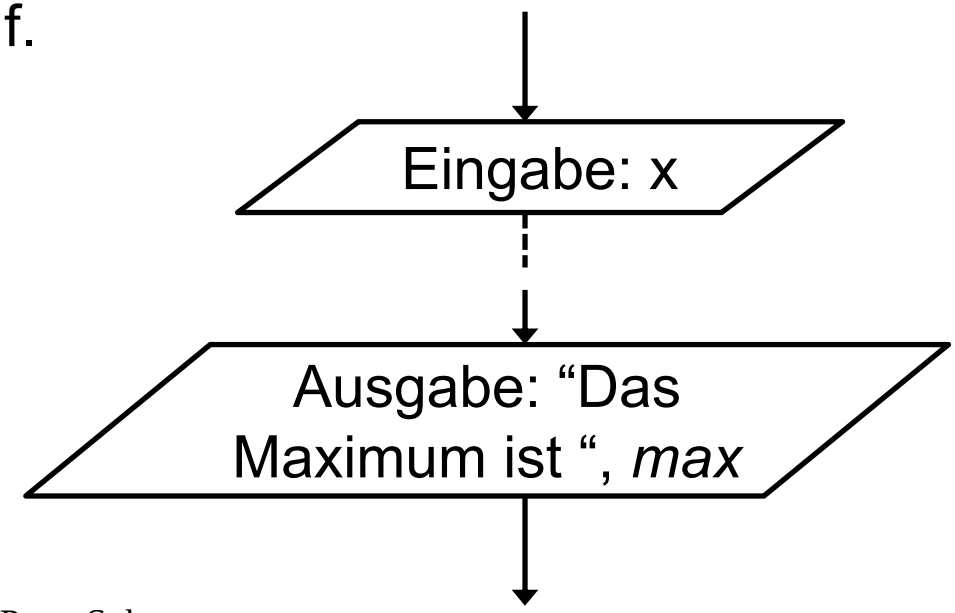
Anschlußstelle (Konnektor)
zwischen zwei
Programmablaufplanteilen

Notation von Anweisungen und Bedingungen für PAP und Struktogramme (1)

Anweisungen:

- Eingabeanweisungen, z.B. Eingabe: *x*
- Ausgabeanweisungen, z.B. Ausgabe: “Das Maximum ist “, *max*

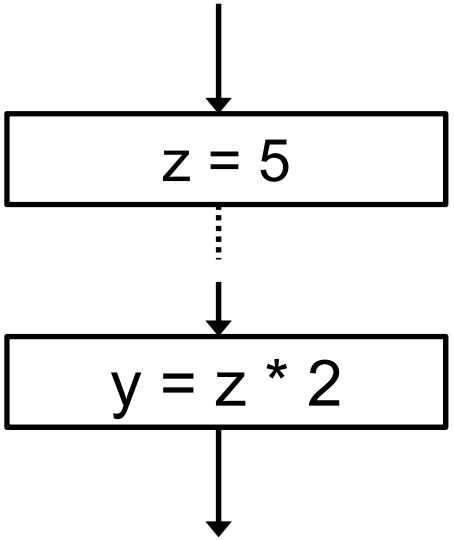
Die Symbole *x* und *max* werden hier für Variablen benutzt. Variablen nehmen Daten während der Abarbeitung des Programms auf.



Notation von Anweisungen und Bedingungen für PAP und Struktogramme (2)

Anweisungen:

- Zuweisungen konstanter Werte, z.B. *z = 5*
- Zuweisungen im Stil mathematischer Ausdrücke, z.B. *y = z * 2.* Hier wird der Variable y der Wert zugewiesen, der sich aus *z*2* errechnet. Allgemein wird der links von = stehenden Variablen der Wert des Ausdrucks rechts zugewiesen.

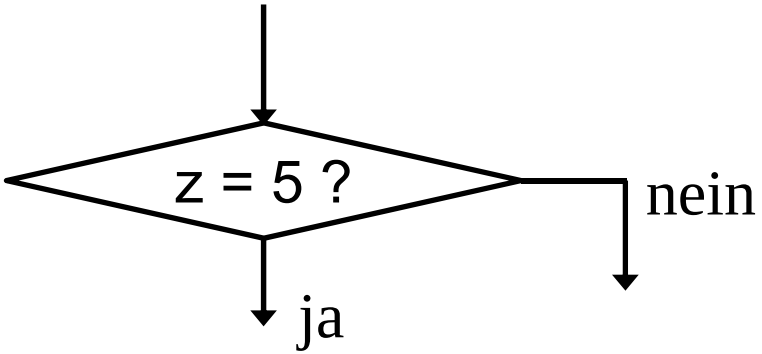


Nicht gültig:
*x+y = 10*z*
Das ist keine Zuweisung, obwohl es ein möglicher mathematischer Ausdruck ist.

Notation von Anweisungen und Bedingungen für PAP und Struktogramme (3)

Bedingungen:

- Als Frage formuliert, ob eine Variable einen bestimmten Wert aufweist, z.B. *z = 0 ?* . Die Bedingung ist dann erfüllt, wenn z den Wert 0 besitzt. Das „?“ kann (muss nicht) verwendet werden.
- Als Gleichung formuliert, z.B. *y = z * z ?* . Wenn die linke und rechte Seite der Gleichung den gleichen Wert ergeben, dann ist die Bedingung erfüllt.
- Als Ungleichung formuliert, z.B. *y > z ?* . Nur bei ungleicher linker und rechter Seite ist die Bedingung erfüllt.



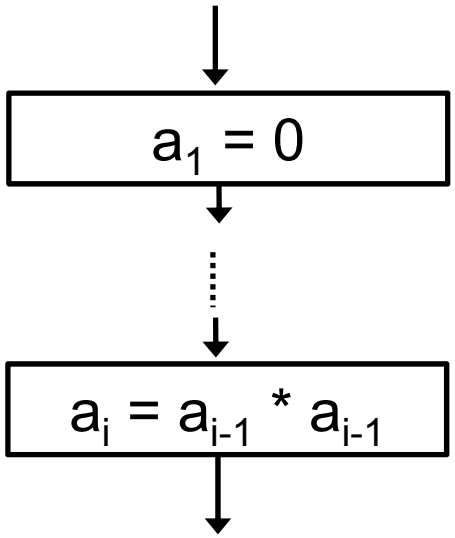
Achtung: Ein Ausdruck wirkt unterschiedlich, je nach dem, ob er in einer Anweisung oder in einer Bedingung verwendet wird.

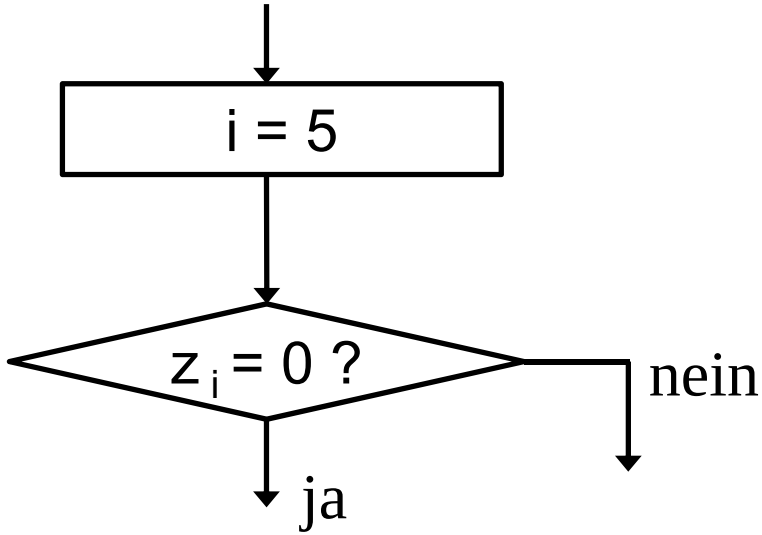
Notation von Anweisungen und Bedingungen für PAP und Struktogramme (4)

Felder:

Algorithmen arbeiten oft auf Feldern, die N einzelne Variable als Folge enthalten, z.B. z_1, z_2, z_3, … z_N

Die einzelnen Elemente können im PAP oder Struktogramm mit Index angegeben werden. Der Index kann ein konkreter Wert sein (z.B. z_1) oder auch als Variable angegeben werden (z.B. z_i).

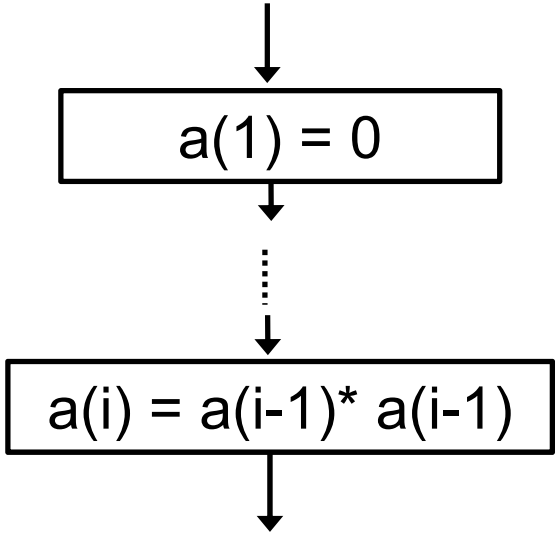


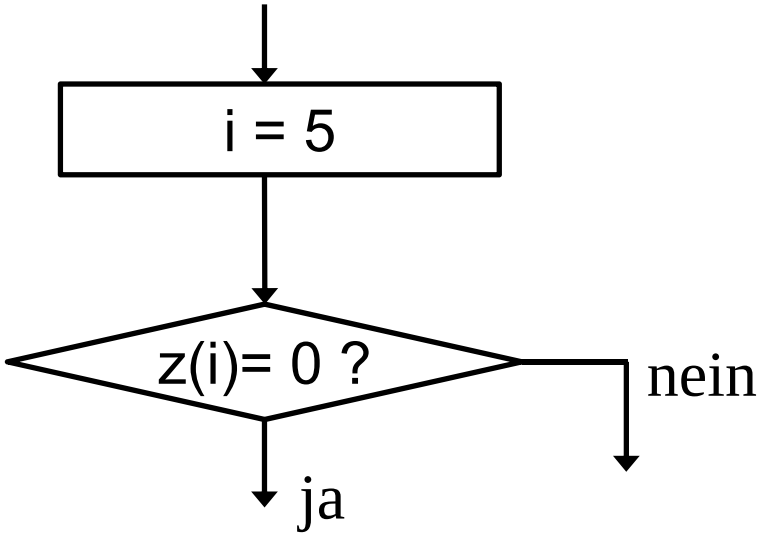


Notation von Anweisungen und Bedingungen für PAP und Struktogramme (5)

Felder (Fortsetzung):

Der Index kann auch mit Indexklammern geschrieben werden, z.B. z(1) oder z(i-1).





Spezielle Algorithmen

Im folgenden Abschnitt sollen ausgewählte Algorithmen, die spezielle Konstrukte des algorithmischen Paradigmas, wie

- Selektion, Mehrfachselektion
- Zyklen verschiedener Formen

enthalten, entwickelt und besprochen werden.

In einem weiteren Abschnitt werden dann Algorithmen mit Prozeduren und Funktionen behandelt und schließlich werden rekursive Algorithmen dargestellt.

Algorithmen mit Selektion

Algorithmen, die nur die Konstrukte Sequenz und Selektion (einschließlich Mehrfachselektion) enthalten sind eigentlich sehr einfache Algorithmen, wenn man einmal ausschließt, dass Prozeduren aufgerufen werden. Die meisten anspruchsvollen Algorithmen enthalten Zyklen.

Beispiel 1:

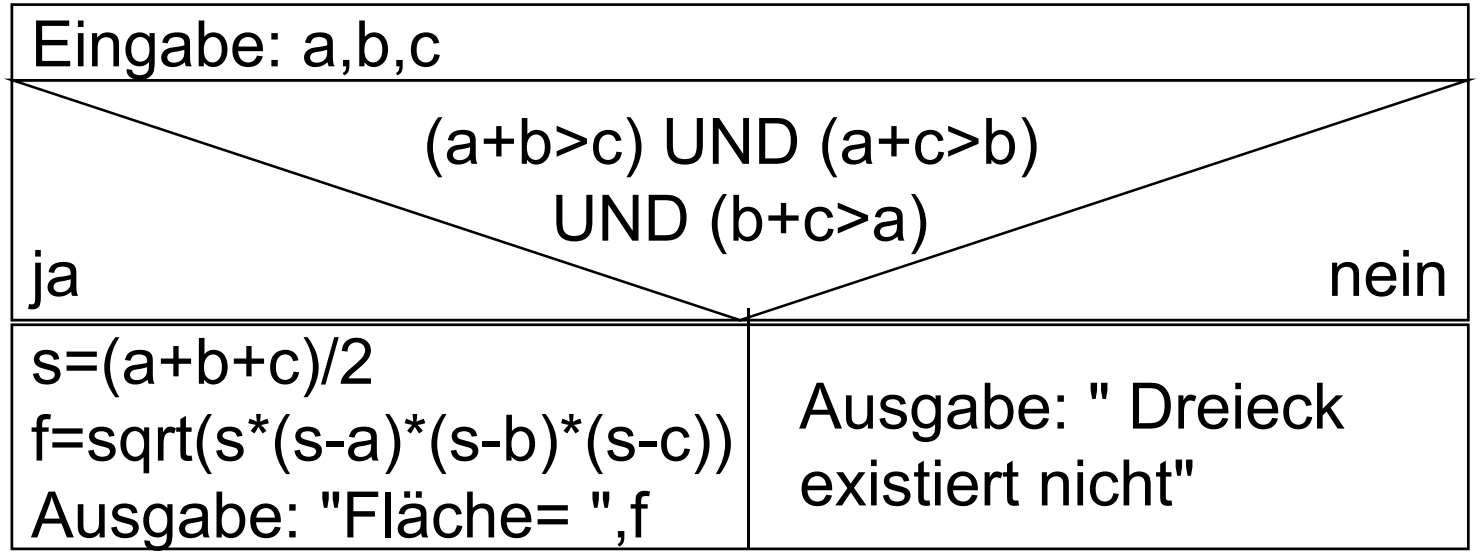
Prüfen der Seitenlängen eines Dreiecks und anschließende Berechnung des Flächeninhalts.
Ein Dreieck mit den Seiten a,b,c hat einen Flächeninhalt

f= $\sqrt{s*(s-a)*(s-b)*(s-c)}$ mit s=(a+b+c)/2

Es ist zu prüfen, ob mit den Seiten wirklich ein Dreieck bildbar ist!

Algorithmen mit Selektion

Beispiel 1: Ein Dreieck kann nur dann gebildet werden, falls a+b>c und a+c>b und b+c>a gilt.
Damit können wir folgenden Algorithmus angeben:



Algorithmen mit Selektion

Beispiel 2:

Ein Algorithmus, der von zwei natürlichen Zahlen x und y ermittelt, ob die eine ein Teiler der anderen ist.

Beispiel 3:

Ein Algorithmus, der ein eingegebenes Datum, bestehend aus den drei Größen t für Tag, m für Monat und j für Jahr, überprüft und ausgibt, ob es sich um ein gültiges Datum aus dem Zeitraum 1583 bis 2100 handelt. Dazu ist das Datum auf einfache Plausibilität zu prüfen, aber auch der Tag in Bezug auf den Monat zu prüfen.

Lösungen zu den Beispielen 2 und 3 werden an der Tafel demonstriert, oder sind Gegenstand der Übung.

Algorithmen mit Selektion und Zyklen

Zyklen in Algorithmen

- werden benutzt, um mehrere Datenelemente (Elemente innerhalb Mengen, Reihen, Feldern, Vektoren, Matrizen) nacheinander zu verarbeiten.

- sind nötig für schrittweise Lösungsverfahren, die sich durch die wiederholte Ausführung einer Regel einer Lösung annähern.

- werden für Verfahren verwendet, die kontinuierlich neue Daten einlesen und verarbeiten

Algorithmen mit Selektion und Zyklen

Beispiel 4: (Basisalgorithmus mit Zählschleife)
Es ist ein Algorithmus zur Bestimmung des Minimums einer Zahlenfolge a_1, a_2 ,... a_n aufzustellen.

Lösung: Mitschrift

Beispiel 5: (Basisalgorithmus mit geschachtelten Zählschleifen)
Es ist eine Weiterentwicklung des Algorithmus 4 vorzunehmen, der eine gegebene Zahlenfolge sortiert, indem er das Minimum bestimmt, an die 1.Position setzt und dann mit dem Rest (ab Position 2) wieder das Minimum bestimmt usw.

Lösung: Mitschrift

Sortieren durch Iteration (1)

Quicksort (als rekursiver Algorithmus) ist ein sehr schneller Sortieralgorithmus, aber ist nicht einfach zu durchschauen.

Im folgenden soll ein sehr einfacher – intuitiver – Sortieralgorithmus angegeben werden.

Bubble Sort

Eingabe: feld a mit Elementen a[1] bis a[n], Feldlänge n
Ausgabe: feld a mit sortierten Elementen

Idee:

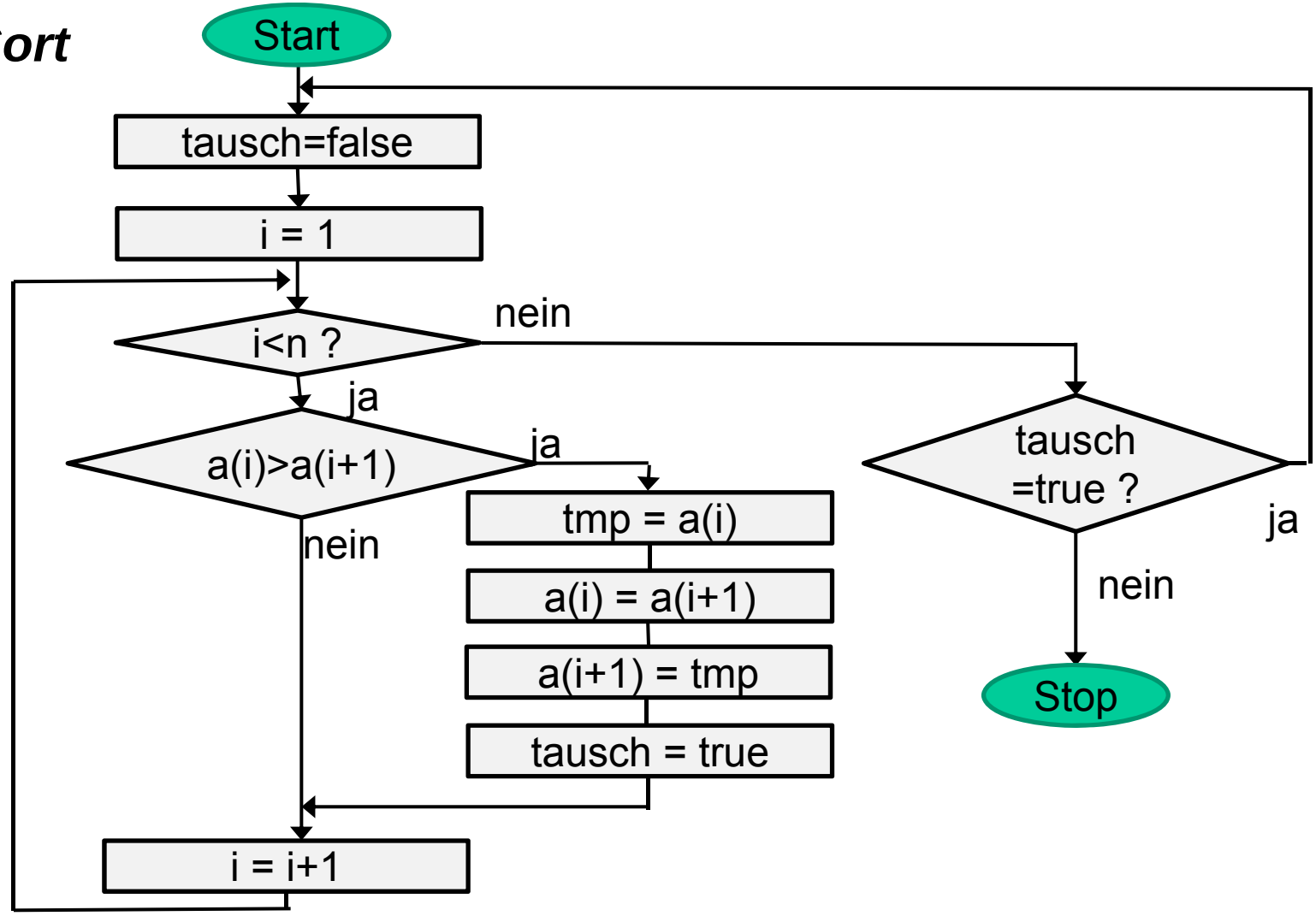
(1) Durchlaufe Feld mit aufsteigendem Index und vertausche benachbarte Felder, falls Sie nicht der geforderten Sortierreihenfolge entsprechen.

Wiederhole (1) solange, bis keine Vertauschungen mehr nötig sind

Sortieren durch Iteration (2)

Bubble Sort

PAP



Rekursive Algorithmen

In der Mathematik sind viele Funktionen rekursiv definiert.

Der Begriff der Rekursion beinhaltet, dass zur Definition einer Funktion diese selbst wieder mit benutzt wird, allerdings mit anderen Argumenten. Eine rekursive Definition benötigt stets eine (nichtrekursive) Anfangs- bzw. Abbruchbedingung.

Beispiel: Fakultät

rekursive Definition : fak(n) = n * fak(n-1)
Anfangsbedingung: fak(0) = 1

Rekursive Algorithmen - Charakteristika

Eine solche Definition ist in der Regel kurz und übersichtlich.

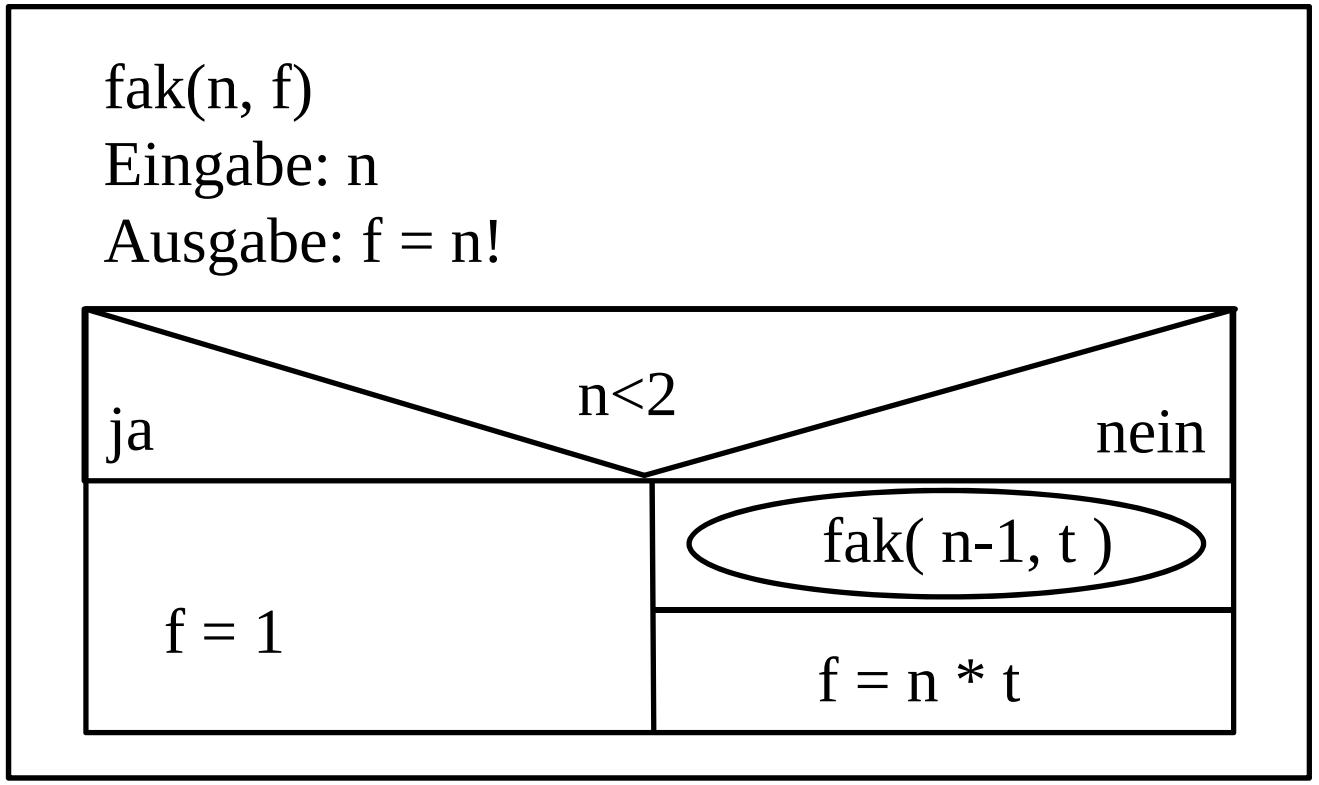
Man erkennt sofort die Grundstruktur des Algorithmus. Eine solche rekursive Definition lässt sich auch sehr leicht unter Verwendung rekursiver Prozeduren (Funktionen) implementieren. Allerdings ist der implementierte Algorithmus häufig ineffizient, d.h. mit einem hohen Ressourcenverbrauch (Speicher und Rechenzeit) verbunden.

Bestimmte rekursive Algorithmen lassen sich in iterative Algorithmen umschreiben und damit effizienter implementieren. Dies trifft insbesondere auf eine sogenannte endständige Rekursion zu, wie z.B. bei der Berechnung der Fakultät.

fak(0)=1; fak(n) = n * fak(n-1);

Rekursive Algorithmen - Fakultät

Das folgende Struktogramm enthält die Umsetzung der rekursiven Definition:

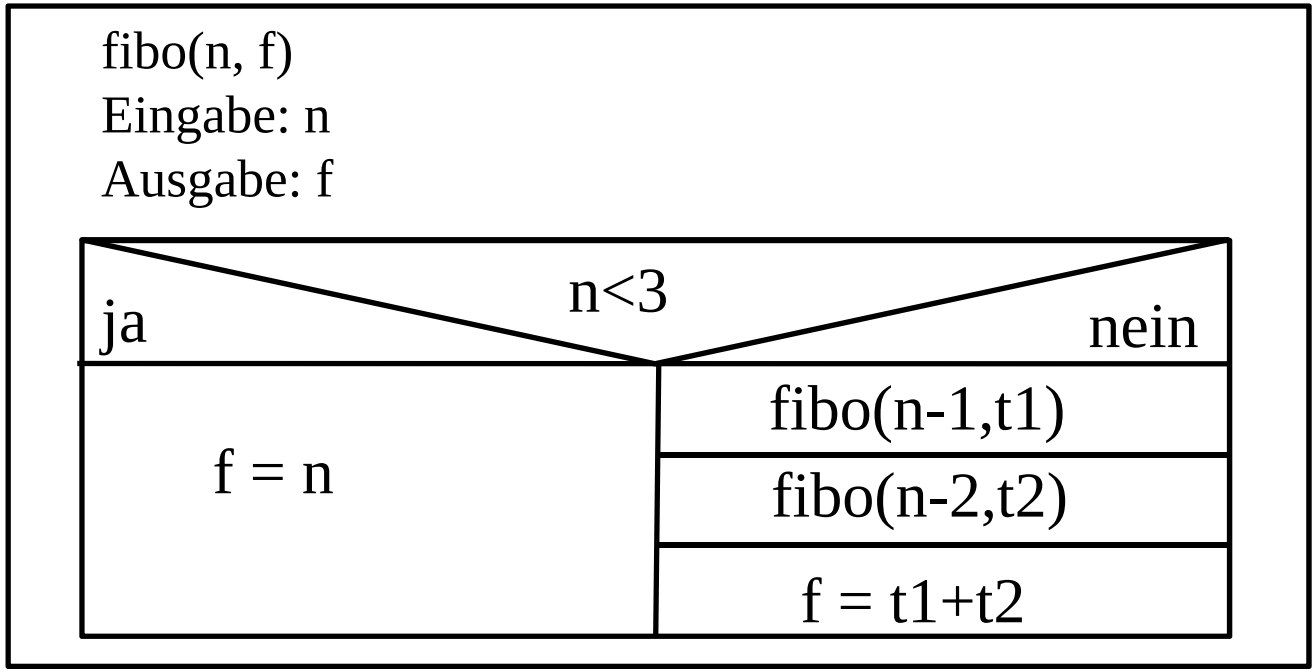


Rekursive Algorithmen – Fibonacci-Zahlen

Rekursive Definition der Fibonacci-Zahlen:
fibo(n) = fibo(n-1) + fibo(n-2)

Anfangsbedingung: fibo(1)=0, fibo(2)=1

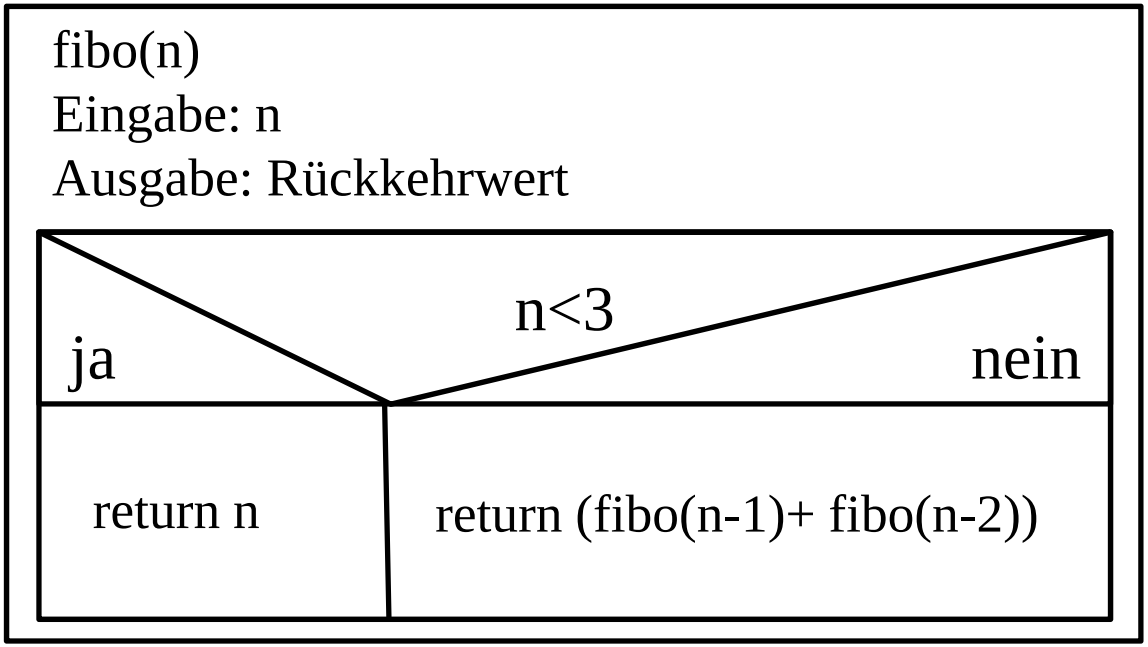
Das folgende Struktogramm enthält die Umsetzung der rekursiven Definition:



Rekursive Algorithmen – Fibonacci-Zahlen

Das folgende Struktogramm benutzt die Rückkehr aus dem Unterprogramm: return

return gibt einen Wert zurück, so dass z.B. x = fibo(5) geschrieben werden kann.



Rekursive Algorithmen – Turm von Hanoi

Das Problem beim Turm von Hanoi besteht in der folgenden Aufgabe:

1. Gegeben ist ein Turm auf einem Standplatz A aus n Scheiben, die übereinander liegen, und zwar immer eine kleinere auf einer größeren Scheibe.
2. Der Turm soll auf einen zweiten Platz B umgesetzt werden, wobei aber beim Umsetzen immer nur eine kleinere auf eine größere Scheibe gelegt werden darf.
3. Bei der Umsetzung darf ein dritter Hilfsplatz C mitbenutzt werden.

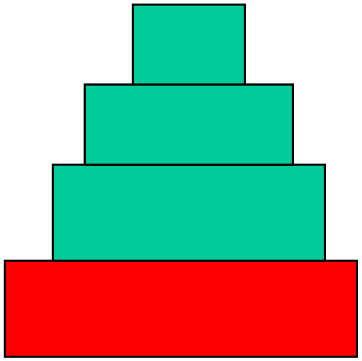
Das C-Programm für dieses Problem wird in der Vorlesung vorgeführt. Es dient als Experimentierprogramm für einen Turm mit einer wählbaren Scheibenanzahl zum Studium der Aufgabenstellung.

Rekursive Algorithmen – Turm von Hanoi

Analysiert man das Problem beim Turm von Hanoi so erkennt man, dass man beim Umsetzen des Turms von n Scheiben vom Platz A zum Platz B erst einmal den Turm von n-1 Scheiben über der größten Scheibe von A nach dem Hilfsplatz C umsetzen muss, um einen Zug der größten Scheibe vom Platz A zum Platz B vornehmen zu können. Danach muss der Turm von n-1 Scheiben vom Platz C wieder auf den Platz B umgesetzt werden.

Rekursive Algorithmen – Turm von Hanoi

Platz A Platz B Platz C

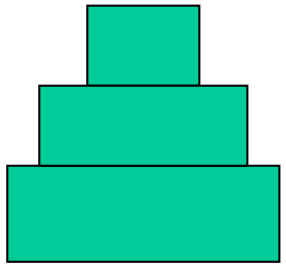


Ausgangssituation

Turm soll nach Platz B umgesetzt werden

Rekursive Algorithmen – Turm von Hanoi

Platz A Platz B Platz C



Turm mit n-1 Scheiben über der größten Scheibe muss auf Hilfsplatz C umgesetzt werden.

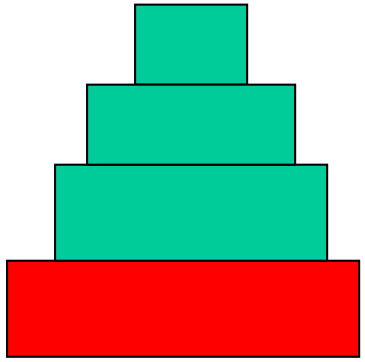
Rekursive Algorithmen – Turm von Hanoi

Platz A Platz B Platz C

Größte Scheibe kann jetzt durch einen Zug vom Platz A zum Platz B befördert werden.

Rekursive Algorithmen – Turm von Hanoi

Platz A Platz B Platz C



Turm mit n-1 Scheiben kann jetzt vom Hilfsplatz C zum Platz B umgesetzt werden.

Rekursive Algorithmen – Turm von Hanoi

Algorithmus rekursiv:

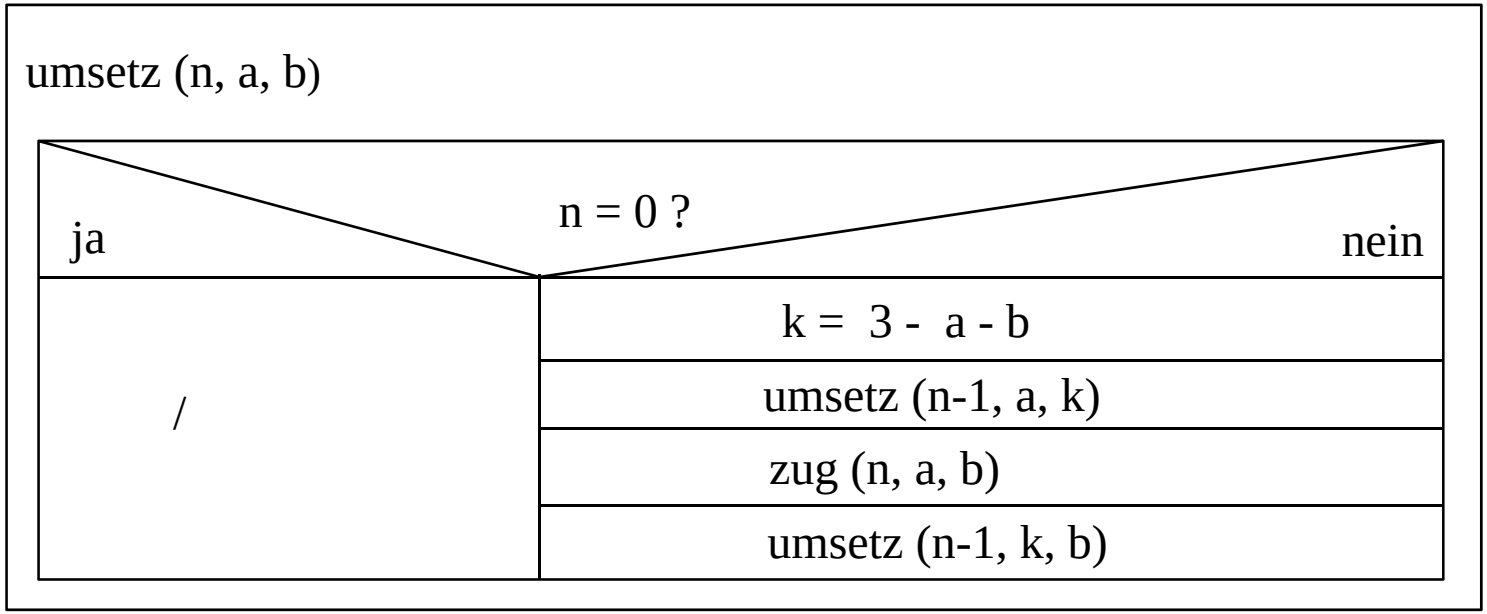
Umsetz(n,A,B) =

Umsetz(n-1,A,C) , *Zug (n,A,B)*, *Umsetz(n-1,C,B)*

Die Rolle des Hilfsplatzes *C* wechselt von Ebene zu Ebene.
Hilfsplatz ist immer der Platz, der in der Umsetzung nicht genannt ist.
Bei *Umsetz(...,A,C)* ist es in der nächsten Ebene der Platz B usw..
Bezeichnet man den Platz *A* mit der Ziffer 0, den Platz *B* mit der Ziffer 1, und den Platz *C* mit der Ziffer 2, so kann der freie Platz immer mit *3-A-B* bezeichnet werden.

Rekursive Algorithmen – Turm von Hanoi

Struktogramm rekursiv:




```
zug( n, a, b)
    Ausgabe: "snr=",n,"von ",p[a],"->",p[b]
```