Teil I

Probleme lösen, Algorithmen, Programmieren

Lernziele

- Begriffe Programm und Algorithmus, Determiniertheit, deterministisch, Finitheit, Terminiertheit kennen
- Für Algorithmen Ein- und Ausgaben spezifizieren können
- Alltägliche Prozesse in Programmablaufplänen, Struktogrammen und Pseudocode formalisieren können

Bestimmen der Wurzel einer Zahl

Einführendes Beispiel

Deklarative Beschreibung: Bestimme x, sodass gilt $x \cdot x = y$.

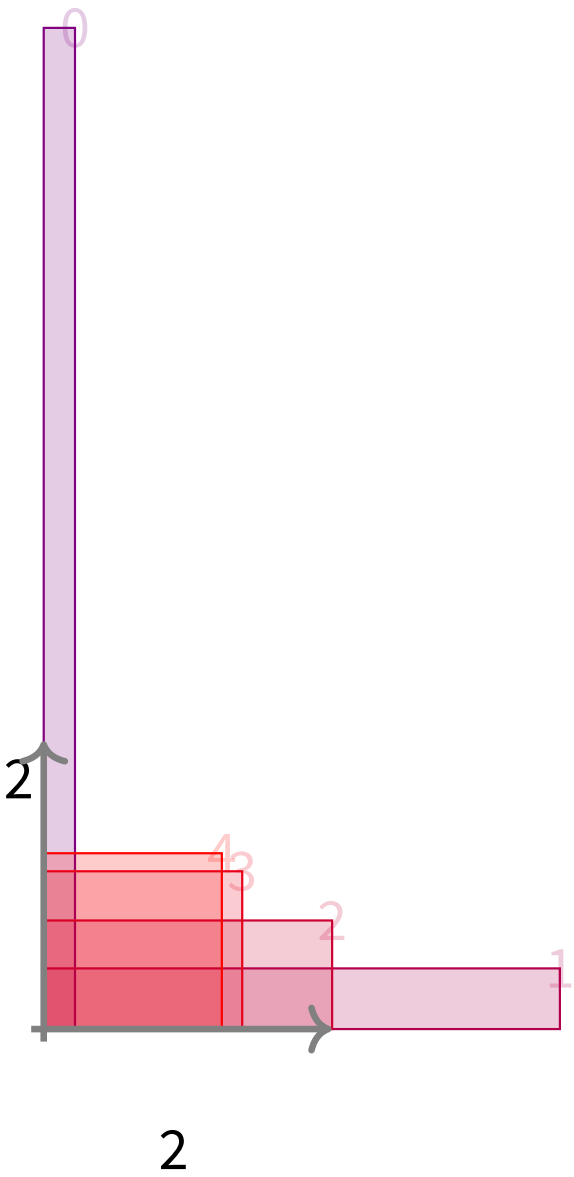
Imperative Beschreibung: (Heron-Verfahren[5]:)

- Wähle eine beliebige Zahl g
- Wenn $g \cdot g$ nicht **nahe bei** y liegt, dann
  - berechne eine neue Zahl aus $(g + \frac{y}{g})/2$
  - Nimm diese neue Zahl als g und wiederhole den Schritt solange $g \cdot g$ nicht nahe bei y liegt

[5] Benannt nach Heron von Alexandria, der es im Werk Metrika beschreibt. Das Verfahren wurde bereits 1000 Jahre zuvor von den Babyloniern benutzt

Visuelle interpretation des Verfahrens

Ziel ist ein Quadrat mit der Fläche y aufzuspannen.
Man wählt deshlab eine beliebige Seitenlänge für die erste Seite und überprüft, ob das Quadrat für beide Seitenlänge gut gewählt ist.
Ist es das nicht, so hat man den Wert z.B. überschätzt. Die Fläche durch die eine Seitenlänge dividiert ergibt die andere Seitenlänge eines Rechtecks mit der Fläche y.
Man sucht also einen besseren Wert, der zwischen den beiden Seitenlängen liegen muss. Ein guter Wert ist hierbei der Mittelwert der beiden Seitenlängen.


0
2
4
3
2
1
2

Wurzel der Zahl 2: Wir nehmen als beliebigen Startwert den Wert 2 an. Dann erhalten wir: $2^2 = 4$, was keine gutes Eregebnis ist.

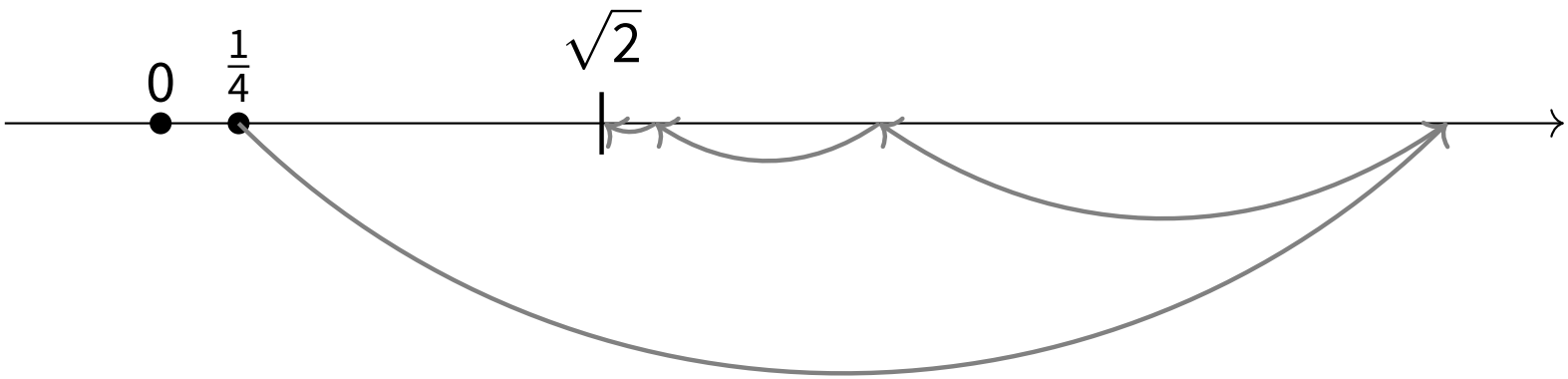
$$g = \frac{1}{2}\left(2 + \frac{2}{2}\right) = \frac{3}{2} = 1,5$$

$$g = \frac{1}{2}\left(\frac{3}{2} + \frac{2}{\frac{3}{2}}\right) = \frac{17}{12} = 1,416666\ldots$$

$$g = \frac{1}{2}\left(\frac{17}{12} + \frac{2}{\frac{17}{12}}\right) = \frac{577}{408} = 1,41421568\ldots$$

Damit wäre eine Annäherung auf fünf Nachkommastellen erreicht

$$\sqrt{2} = 1,41421356\ldots$$



Was ist ein Algorithmus?

Zu dem Begriff **Algorithmus** gibt es keine klare Definition. Es gibt jedoch Beschreibungen, die auf Begriffen basieren, die nur vage definiert sind.

Algorithmus

Ein **Algorithmus** ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten.[a]

[a] Die unterstrichenen Wörter sind nicht näher definiert, wodurch diese Beschreibung nicht als formale Definition herhalten kann.

Somit können sie zur Ausführung in einem Computerprogramm implementiert, aber auch in menschlicher Sprache formuliert werden. Bei der Problemlösung wird eine bestimmte Eingabe in eine bestimmte Ausgabe überführt.[6]

[6] wp

Algorithmus (alternativ)

Ein **Algorithmus** ist ein Verhaltensmuster, das eine Menge auszuführender Aktionen (Handlungen) und deren zeitliche Abfolge festlegt.

Herkunft des Wortes „Algorithmus“

- Abu Ja’far Muhammad ibn Musa **Al-Khwarizmi**
  - ca. 780–850
  - geb. im heutigen Usbekistan
  - choresmischer Mathematiker, Astronom und Geograph
- Werke im „Haus der Weisheit“ (Bagdad)
  - Mathematiker, Astronom und Geograph
  - beteiligt an der Übersetzung der Werke griechischer Mathematiker ins Arabische
- Schrieb: „Über das Rechnen mit indischen Ziffern“
  - Rechnen im Dezimalsystem
  - Lateinische Fassung: „Algorismi de …“ („Das Werk des Al-Khwarizmi über …“)

Abb. Bild von Al-Khwarizmi auf einer sowjetischen Jubiläumsbriefmarke

Was ist ein Programm?

Definition 1 (Programm). Ein **Programm** ist ein streng formalisierter, eindeutiger und detaillierter Algorithmus, der maschinell ausgeführt werden kann.[a].

[a] nach Prof. B. Jung, Grundlagen der Informatik, Skript WS2007/2008

Programmiersprache

Definition 2 (Programmiersprache). Eine **Programmiersprache**[a] ist eine formalisierte Sprache zum (auf-/be-)schreiben von Algorithmen, die ausgeführt werden sollen.

[a] die im Rahmen dieser Vorlesung erlernte Programmiersprache heißt C++. Weiter im Studium werden Ihnen zumindest Java, Haskell, Pyhton, JavaScript und C begegnen.

Das Programm zur Berechnung der Wurzel nach dem Heron-Verfahren könnte in C++ wie folgt aussehen:

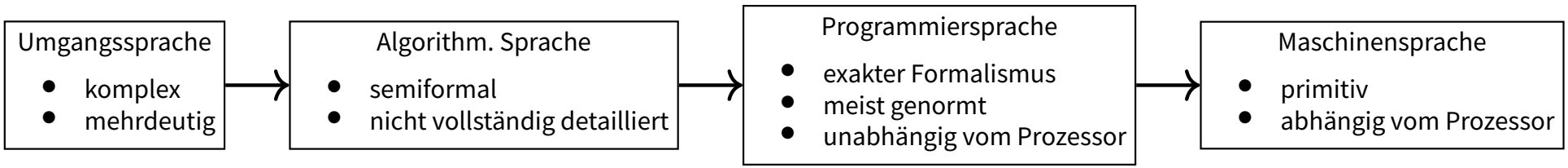
Heron-Verfahren

```
1  #include <iostream>
2
3  int main()
4  {
5      double y;
6      std::cout << "Von welcher Zahl soll die Wurzel bestimmt werden? "
7      std::cin >> y;
8      double g = 1.0;
9      while( std::abs( g * g - y ) > 1e-9 )
10     {
11         std::cout << "g=" << g << "\n";
12         g = ( g + y / g )/2.0;
13     }
14     std::cout << "Die Wurzel ist " << g << std::endl;
15 }
```

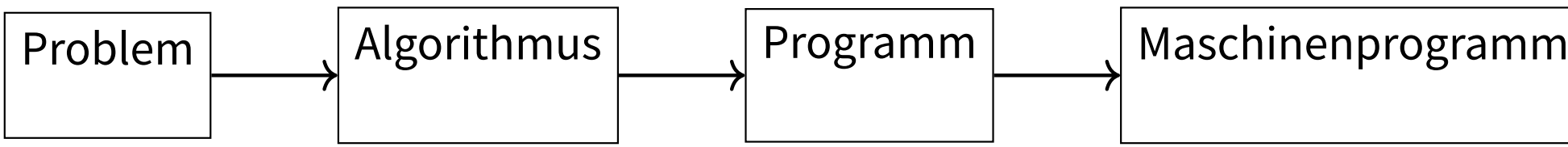
Programmieren

Vom Problem zum Maschinenprogramm

Tab. Die Entstehung eines Programms



Tab. Übergang vom Problem/der Aufgabenstellung zum Maschinenprogramm



Zur Unterstützung der Entwicklung von Programmen gibt es verschiedene Hilfsmittel. Für die Formalisierung eines Programms verwenden wir hier:

Pseudocode eine formalisierte Version der Alltagssprache

Programmablaufpläne Grafische Darstellung des Ablaufs eines Programms

Struktogrammen Übersetzung der Programmablaufpläne in eine sequenzielle Darstellung (von oben nach unten zu lesen)

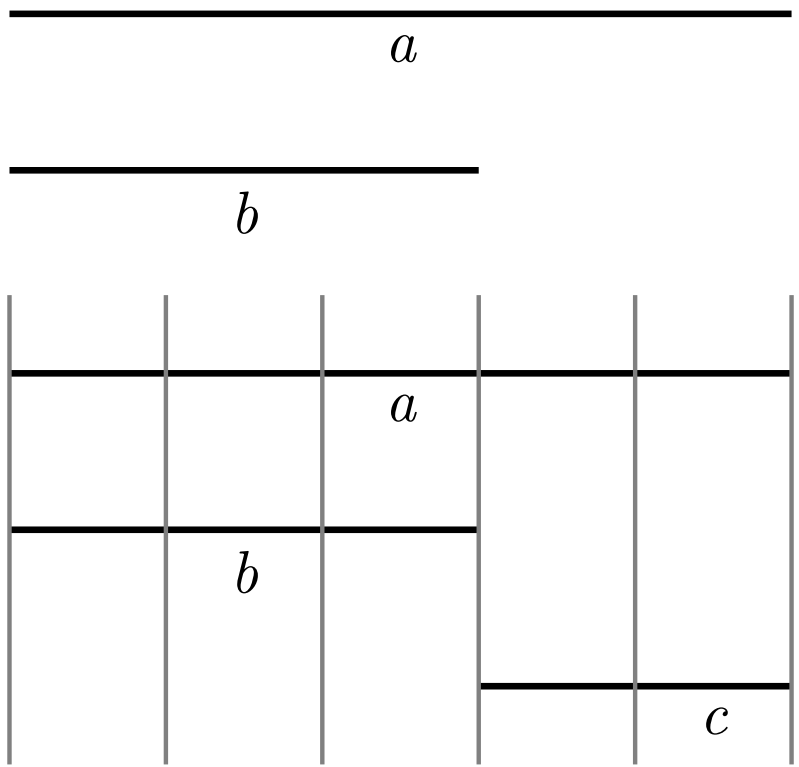
Beispiel für Pseudocode

Eingabe : $a \in \mathbb{N}$
Ausgabe : die Wurzel der Zahl a
$g \leftarrow$ positive Zufallszahl
solange $g^2 \neq a$ **tue**
  $g \leftarrow \frac{1}{2}\left(g + \frac{a}{g}\right)$
Ende
Ergebnis : a

Algorithmus 1 : Algorithmus zur Bestimmung der Wurzel der positiven Zahl a

Beispiel für Pseudocode

Bestimmung des größten gemeinsamen Teilers (ggT) zweier Zahlen:
Seien a und b zwei Zahlen, $a, b \in \mathbb{N}$.



Eine Zahl, die a teilt und gleichzeitig b teilt muss auch c teilen.

Diese Eigenschaft können wir nutzen, um einen Algorithmus zu formulieren.

Eingabe : $a, b \in \mathbb{N}$
Ausgabe : der ggT der Zahlen a und b
solange $a \neq b$ **tue**
  wenn $a \geq b$ **dann**
    $a \leftarrow a - b$
  sonst
    $b \leftarrow b - a$
  Ende
Ende
Ergebnis : a

Algorithmus 2 : Algorithmus für ggT(a,b)

Beispiel 3 (Addition zweier nichtnegativer Ganzzahlen). Eine halbwegs formale Darstellung der Addition zweier positiver Ganzzahlen könnte wie folgt aussehen:

Algorithmus : Schriftliche Addition

Eingabe : Zahlen a, b, definiert über Ziffernfolgen (Zehnersystem)
Ausgabe : Summe c der Zahlen mit $c = a + b$
Ergänze führende Nullen, sodass die Ziffernfolgen gleich lang sind;
Merke Übertrag von 0;
solange Ziffern vorhanden, nimm unbearbeitete Ziffern rechts in der Ziffernfolge, tue
    Addiere die zwei Ziffern und den Übertrag und merke das Ergebnis;
    Falls das Ergebnis größer als 9 ist, merke eine 1 im Übertrag, ansonsten eine 0;
    Füge letzte Ziffer des Ergebnis als neue Ziffer links der Summe hinzu;
Falls Übertrag nicht null, füge Übertag der Summe links hinzu;
Interpretieren Ziffernfolge im Summe (Zahl im Zehnersystem);

Folgende Aussagen treffen auf diesen Algorithmus zu:

- Das Ergebnis ist hier immer **eindeutig** und **wohldefiniert**.
- Dieser Algorithmus macht bei gleicher Eingabe in jedem Schritt exakt das gleiche (er ist **deterministisch**).
- Dieser Algorithmus besitzt sich wiederholende Anweisungen (einen **Zyklus**).

Schleife

Der Begriff **Zyklus** (auch **Schleife**, Wiederholung, Loop) in der Programmierung bezeichnet die wiederholte Ausführung von Teilen eines Programms. Eine Schleife hat eine **Schleifenbedingung**, die angibt, wie oft oder bei welcher Bedingung die Schleife ausgeführt werden soll.[7] Wenn man Programme mit Schleifen schreibt oder untersucht, so definiert man sich **Schleifeninvarianten**.[8] Das sind Bedingungen, die nach jeder Ausführung der Schleife gelten.

[7] In dem vorhergehenden Fall ist die Bedingung „Ziffern vorhanden?“

[8] Hier wäre die Schleifeninvariante: Alle abgearbeiteten Zahlen mit dem vorangestellten Übertrag ergeben die Summe der bisher bearbeiteten Zahlen.

Was ist eigentlich ein Computer?

- engl. Computer, to compute, Rechnen
- deutsch Rechner
- franz. ordinateur, Ordnen, zählen..

Eigentlich sind sie Datenverarbeiter. Besser: Richard Feynman:
`https://www.youtube.com/watch?v=EKWGGDXe5MA`

Für uns wichtige Bestandteile

Wir abstrahieren hier:

- Speicher, das Gedächtnis. Jeder Eintrag im Speicher hat ein Bezeichnung (eine Adresse) wo er zu finden ist.
- Zentralprozessor (CPU). Hier werden Befehle ausgeführt.
- Ein- und Ausgabegeräte. Hier werden Daten entgegengenommen oder Ergebnisse ausgegeben. Diese können teilweise als Speicher abstrahiert werden und an anderer Stelle gelesen werden.

Betriebssystem

- Software/Programm, die zwischen einem Programm/einer Applikation und der Hardware Informationen austauscht.
- Benutzerschnittstelle (User Interface): Oft Teil des Betriebssystems, das dem Benutzer (also uns) erlaubt, Befehle an das Betriebssystem oder das Programm zu senden.

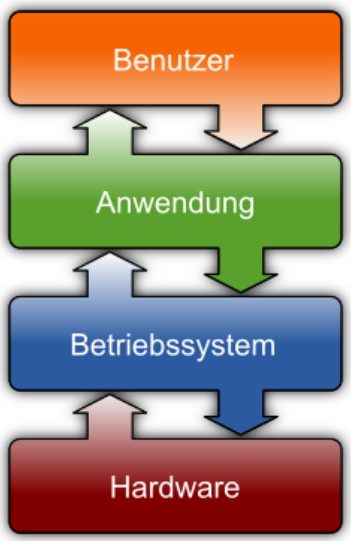


Abb. Quelle: WP

Satz von Böhm-Jacopini

Strukturiertes Programmieren

Satz 4 (Böhm-Jacopini). Jedes Programm kann auf der untersten Ebene beschränkt werden auf lediglich drei Kontrollstrukturen:

1. Sequenz
   (Hintereinander auszuführende Programmanweisungen),
2. Auswahl/Selektion
   (Verzweigung anhand einer Bedingung),
3. Wiederholung/Iteration
   (Schleifen, Zykel von Unterprogrammen mit einer Abbruchbedingung).

Vorgehen beim Schreiben von Algorithmen:

$$\text{Problem} \xrightarrow{\text{Erfahrung+Kreativität}} \text{Algorithmus} \xrightarrow[\text{Maschine}]{\text{Handwerk}} \text{Programm} \xrightarrow{\text{Compiler}}$$

Diese Vorlesung vermittelt:

- das übliche Handwerkszeug sowie
- Routineverfahren zur Problemlösung.

Sie ersetzt nicht die Erfahrung und ihre eigene Kreativität. Beides bekommt man durch

- viel, viel, viel Übung,
- Lesen von (gutem) fremden Code wie ein gutes Buch
  - KDE, gnome, Teile des Linux Kernels (Vorsicht C, nicht C++)
- Lernen von besonders schlechtem fremden Code.

Determiniertheit

Definition 5 (Determiniertheit). Ein Algorithmus ist **determiniert**, wenn dieser bei jeder Ausführung mit gleichen Startbedingungen und Eingaben gleiche Ergebnisse liefert.

Deterministisch

Definition 6 (Deterministisch). Ein Algorithmus ist **deterministisch**, wenn zu jedem Zeitpunkt der Algorithmusausführung der nächste Handlungsschritt eindeutig definiert ist.

Ist der Algorithmus nicht deterministisch, so spielt in mindestens einem Schritt der Zufall eine Rolle. Ein nicht deterministischer Algorithmus kann trotzdem determiniert sein, somit muss ein determinierter Algorithmus nicht zwangsläufig deterministisch sein.

Finitheit

Definition 7 (Statische Finitheit). Ein Algorithmus ist **statisch finit**, wenn seine Beschreibung eine endliche Länge besitzt.

Das heißt, dass der Quelltext muss also aus einer begrenzten Anzahl von Zeichen bestehen.

Definition 8 (Dynamische Finitheit). Ein Algorithmus ist **dynamisch finit**, dann darf er zu jedem Zeitpunkt seiner Ausführung nur begrenzt viel Speicherplatz benötigen.

Es erscheint offensichtlich, dass ein implementierbarer Algorithmus diese beiden Eigenschaften erfüllen muss um realisierbar zu sein.

Terminiertheit

Definition 9 (Terminiertheit). Ein Algorithmus **terminiert überall** oder **ist terminierend**, wenn er für jede mögliche Eingabe nach endlich vielen Schritten anhält (oder kontrolliert abbricht).

Ein nicht-terminierender Algorithmus (somit zu keinem Ergebnis kommend) gerät (für mindestens eine bestimmte Eingabe) in eine so genannte Endlosschleife.

Computational Methods

Für manche Abläufe ist ein nicht-terminierendes Verhalten gewünscht: z. B. Steuerungssysteme, Betriebssysteme und Programme, die auf Interaktion mit dem Benutzer aufbauen. Solange der Benutzer keinen Befehl zum Beenden eingibt, laufen diese Programme beabsichtigt endlos weiter.
Donald E. Knuth[9] schlägt in diesem Zusammenhang vor, nicht terminierende Algorithmen als rechnergestützte Methoden („**Computational Methods**“) zu bezeichnen.
Darüber hinaus ist die Terminierung eines Algorithmus (das Halteproblem) nicht entscheidbar. Das heißt, das Problem, festzustellen, ob ein (beliebiger) Algorithmus mit einer beliebigen Eingabe terminiert, ist nicht durch einen Algorithmus lösbar.

[9] Amerikanischer Informatiker (1938-)

Computerprogramm

Definition 10 (Computerprogramm). Ein **Computerprogramm** oder kurz **Programm** ist eine den Regeln einer bestimmten **Programmiersprache** genügende Folge von Anweisungen (bestehend aus Deklarationen und Instruktionen), um bestimmte Funktionen bzw. Aufgaben oder Probleme mithilfe eines Computers zu bearbeiten oder zu lösen.

Ein Programm ist also zum Beispiel die Umsetzung eines Algorithmus in einer konkreten Programmiersprache.

Programmiersprachen sind formale Sprachen zur Formulierung von Datenstrukturen und Algorithmen, die automatisiert übersetzt werden können, sodass sie von einem Computer ausgeführt werden können.
Wie gesprochene Sprachen verfügen sie über

- eine **Syntax**, die regelt, welche Symbole aus Zeichen zusammengesetzt werden können;
- eine **Grammatik**, die regelt, welche Kombinationen von Symbolen Sinn ergeben;
- eine **Semantik**, die die Bedeutung der Abfolge von Symbolen regelt.[10]

[10] Mehr zu den einzelnen Punkten später im Studium

Abb. Ada Lovelace 1836, Gemälde von Margaret Sarah Carpenter

Augusta Ada Byron King, Countess of Lovelace, kurz **Ada Lovelace**[11] wird das erste veröffentlichte **Programm** für einen nie fertiggestellten mechanischen Computer zugeschrieben, das viele Aspekte der heutigen Programmierung enthielt.

[11] britische Mathematikerin, 1815-1852

Abb. Konrad Zuse, 1992

Konrad Ernst Otto Zuse: deutscher Bauingenieur, Erfinder, Unternehmer (1910-1995), Entwickler des Z3 (1941, zusamen mit Helmut Schreyer), des ersten funktionstüchtigen, vollautomatischen, vollprogrammierbaren, programmgesteuerten und frei programmierbaren Rechners.

Abb. Grace Hooperwp

Die Idee, Computerprogramme in einer „verständlichen“ Sprache zu verfassen wird **Grace Brewster Murray Hooper**[12] zugeschrieben. Sie entwickelte 1952 den ersten Übersetzer (Compiler).

[12] US amerikanische Informatikerin, 1906–1992

Maschinensprachen

- Low Level
- maschinenabhängig
- dargestellt als
- Zahlenfolgen oder Abkürzungen
- vom Rechner direkt verarbeitbar
- vom Menschen schlecht lesbar
- Aber: Unmittelbare Kontrolle über alles, was geschieht!

```
0010 0001 0000 0100
0001 0001 0000 0101
0011 0001 0000 0110
0111 0000 0000 0001
0000 0000 0101 0011
1111 1111 1111 1110
0000 0000 0000 0000
```

Assemblersprachen

- „Middle Level“
- maschinenabhängig
- Verwendung mnemontechnischen Abkürzungen für elementare Maschinenbefehle
- relative Addressierung
- nur nach Übersetzung in Maschinensprache verwendbar
- immer noch exakte Kontrolle über alles, was geschieht

Assembler

```
1  LDA A
2  ADD B
3  STA C
4  HLT
5  A, DEC 83
6  B, DEC -2
7  C, DEC 0
8  END
```

Der Assemblerquelltext wird mit einem Programm (dem Assembler) in

Höhere Programmiersprachen

- Beschreiben Probleme/Anweisungne auf höherem Niveau
- häufig problemorientierte Sprachkonstrukte
- rechnerunabhängig
- vor Ausführung Übersetzung erforderlich
- gut lesbar

Höhere Programmiersprache: C++

```
1  c = 83 + (-2);
```

Der Quelltext wird mit einem Programm (dem Compiler) in Assembler oder Maschinencode übersetzt.

Beispiele für höhere Programmiersprachen I

FORTRAN (FORmula TRANslator)

- seit 1954 zunächst von IBM entwickelt
- Einsatzgebiet: technisch-naturwissenschaftliche Anwendungen

COBOL (COmmon Business Oriented Languate)

- Einsatzgebiet: kommerziell-administrative Anwendungen
- 1999 waren mehr als 50% der kommerziellen Software in COBOL

C

- Weiterentwicklung der Sprache B und BCPL
- C ist Entwicklungssprache für viele Betriebssysteme (UNIX, Linux, FreeBSD, Windows)

Beispiele für höhere Programmiersprachen II

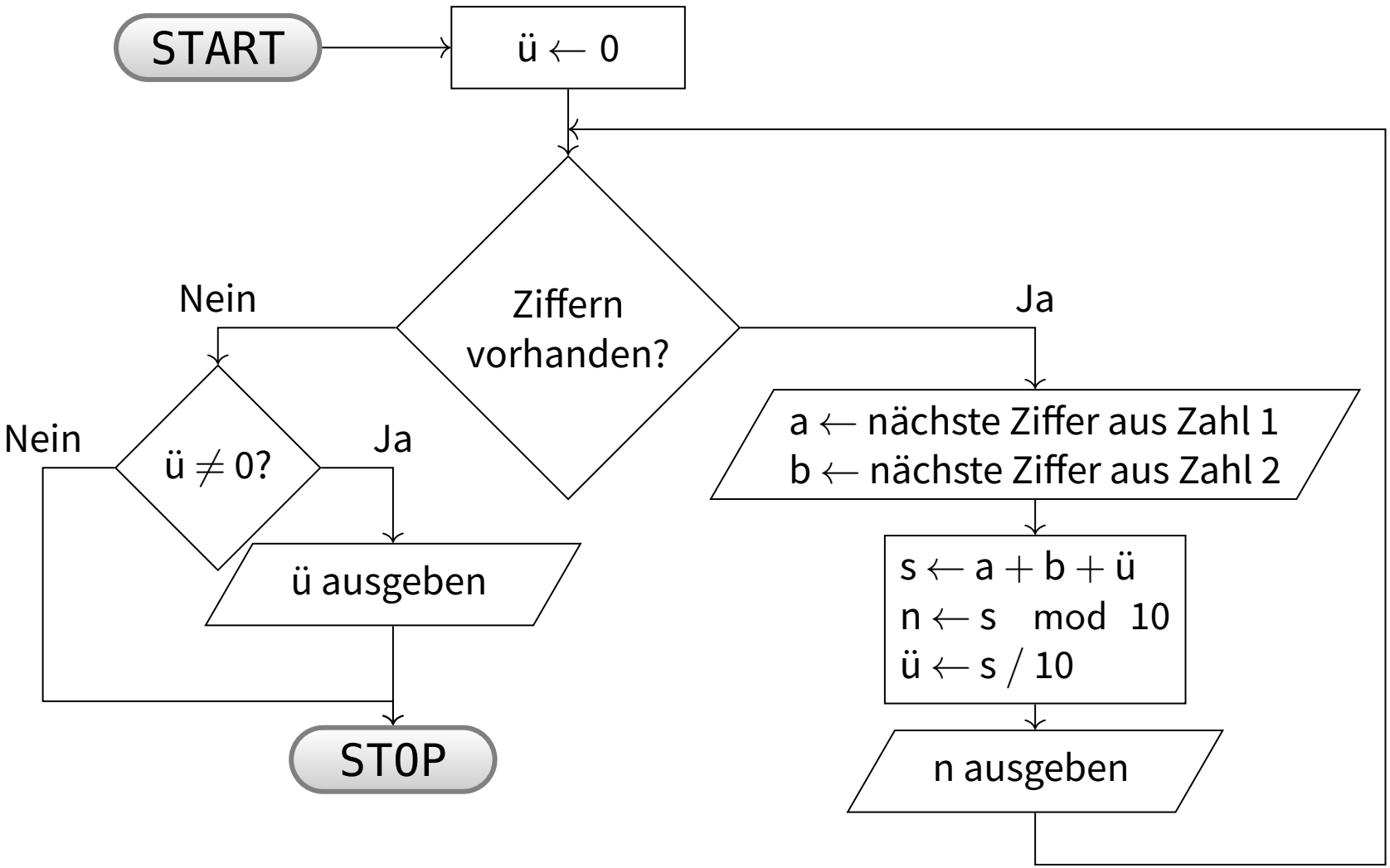
C++

- Erweiterung von C um objektorientierte Konstrukte
- Einfluss der Sprache Simula

Java

- eingeführt von SUN Microsystems 1995
- syntaktische Ähnlichkeit zu C und C++
- Plattformunabhängige Binärdateien

Programmablaufplan (PAP)



Programmablaufplan I

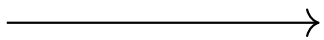
- Ablaufpläne beschreiben welche Anweisungen in welcher Reihenfolge ausgeführt werden müssen.
- Sie verfügen über einen klar definierten Start und Endpunkt
- Einzelne Abläufe (Prozesse) können dabei wieder durch Programmablaufpläne dargestellt werden bis sie vollständig durch triviale Anweisungen beschrieben sind.

Programmablaufplan II

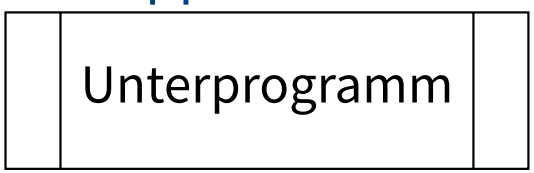
Flussdiagramme bestehen nach DIN 66001 (1966) aus folgenden häufig verwendeten Symbolen:

Kreis, Oval, Runde Ecken Kontrollpunkt

Pfeil, Linie Verbindung zum nächsten Element

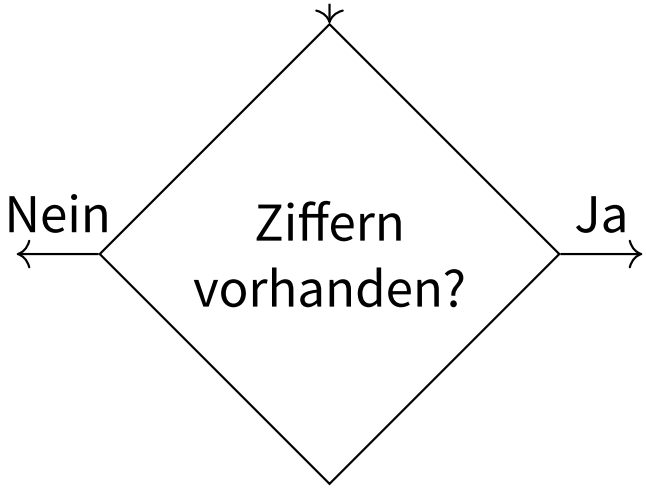


Rechteck mit doppelten Linien Unterprogramm, Komplexe Anweisung



Programmablaufplan III

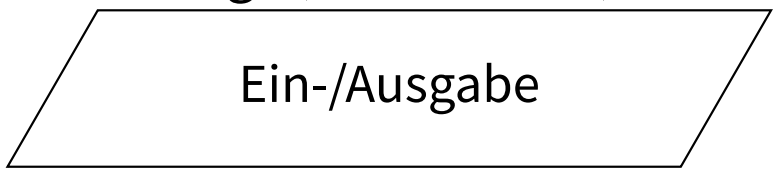
Raute Verzweigung, Entscheidung



Rechteck Operation, Tätigkeit (einfache Anweisungen)

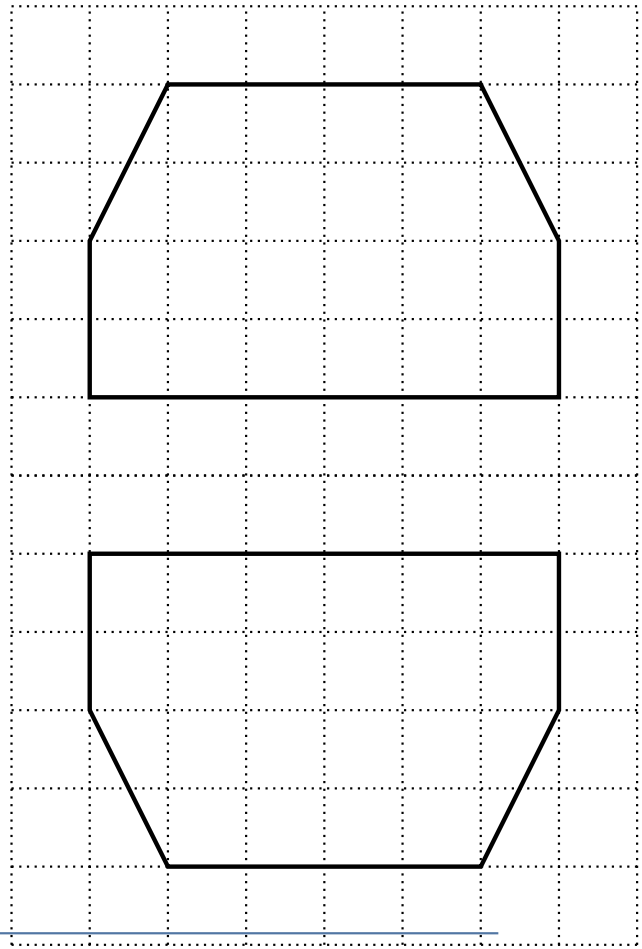
Programmablaufplan IV

Parallelogramm (1966 spezifiziert, 1982 entfernt) Ein-/Ausgabe (z.B. Datenträger, Bildschirm, Netzwerk)



Programmablaufplan V

Schleifenbegrenzung (1982 spezifiziert)[13]



Schleifenanfang

Schleifenende

[13] nach: web.archive.org: htw-berlin, GdP

Struktogramme

Programmablaufpläne werden schnell unübersichtlich. Bei der Programmierung versucht man deshalb, Teile der Ablaufpläne in Blöcke zu gliedern.

- Dabei fasst man als kleinesten Block einzelne Anweilsungen zusammen.
- Aus diesen baut man als nächsten Block Anweisungsfolgen auf, die streng hintereinander ausgeführt werden.
- Verzweigungen werden nach ihrem Sinn in spezielle Blöcke gegliedert.

Ein Beispiel für eine Darstellung, die sich näher an den Programmen orientiert sind die 1972 eingeführten Nassi-Shneiderman-Diagramme wp [14], die häufig als **Struktogramme**[15] bezeichnet werden.
Sie eignen sich besser, einen Programmablauf „von oben nach unten“ darzustellen.

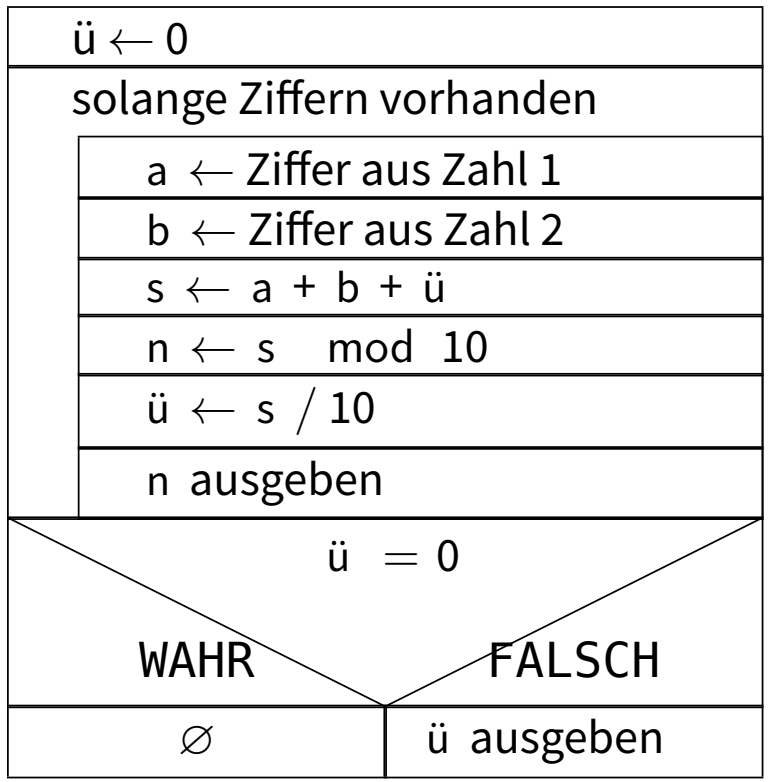
[14] Benannt nach Isaac Nassi wp (1949) und Ben Shneiderman wp (1947), US-amerikanische Informatiker

[15] standardidsiert in DIN 66261 (Überblick bwi haw-hamburg)

Ihr Bausteine sind **Blöcke**, die durch spezielle Operationen verschachtelt werden können. Die einfachste Version eines Blocks ist eine **Anweisung** oder ein Folge von Anweisungen, die wir **Sequenz** nennen.

Struktogramme

Struktogramme bieten eine alternative Möglichkeit, Algorithmen darzustellen. Sie sind näher am späteren Programmquelltext, da sie eine klare lineare Struktur haben.



Elemente von Struktogrammen I

Struktogramme bestehen aus folgenden Elementen:

Anweisung

Berechne die Kreisfläche

Anweisungsfolge

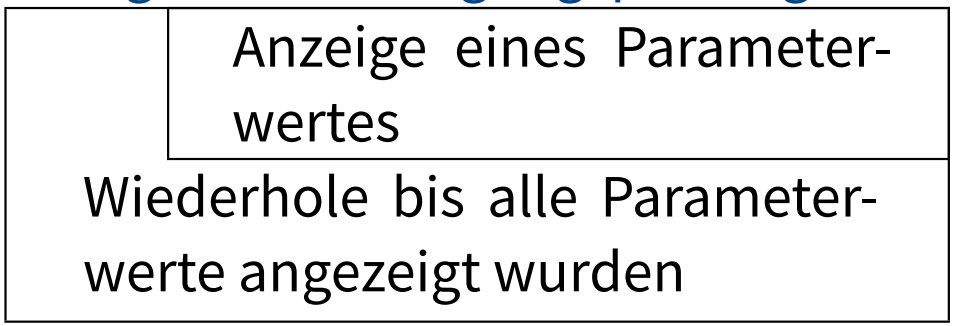
Eingabe Radius
Berechnung Durchmesser

Zyklus mit vorausgehender Bedingungsprüfung

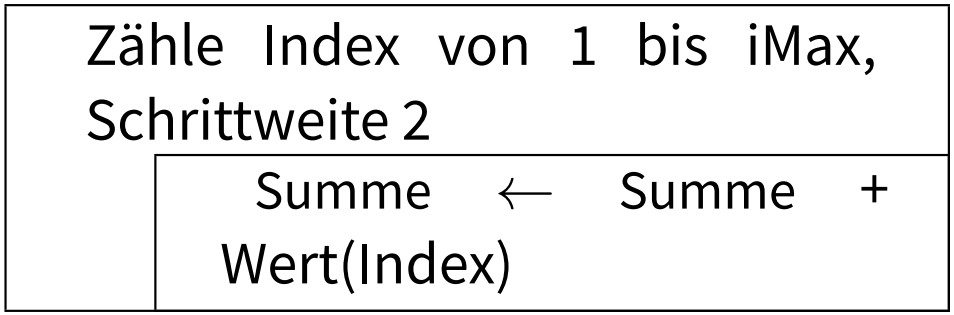
Wiederhole bis Text keine doppelten Leerzeichen enthält	
	Ersetze doppelte Leerzeichen durch einfache Leerzeichen

Elemente von Struktogrammen II

Zyklus mit nachfolgender Bedingungsprüfung

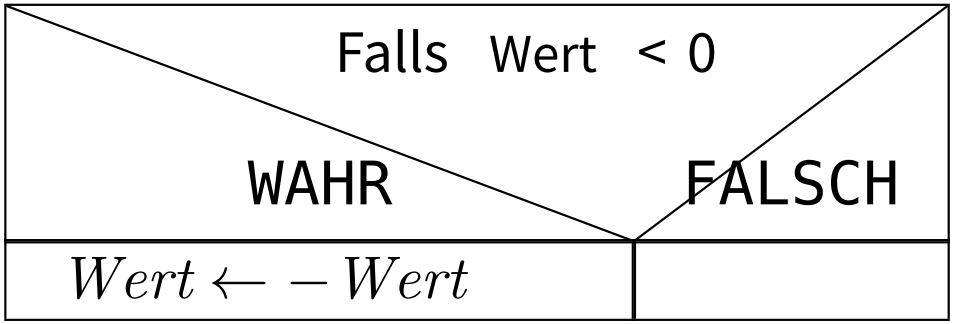


Zählergesteuerte Schleife

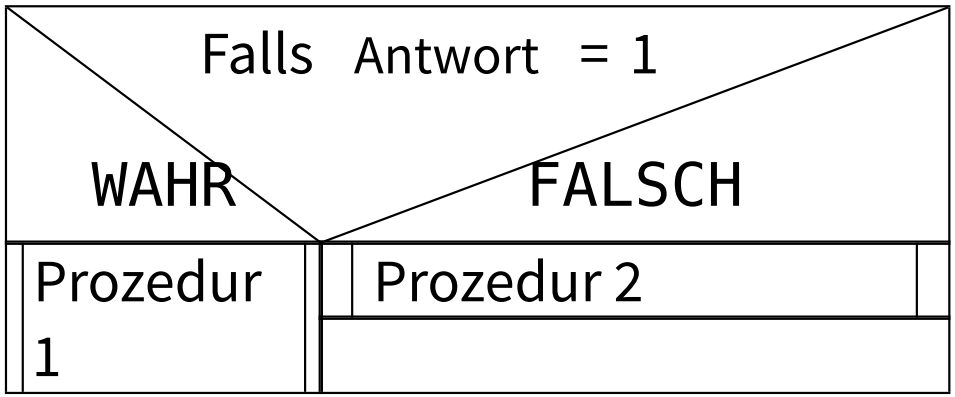


Elemente von Struktogrammen III

Bedingte Anweisung

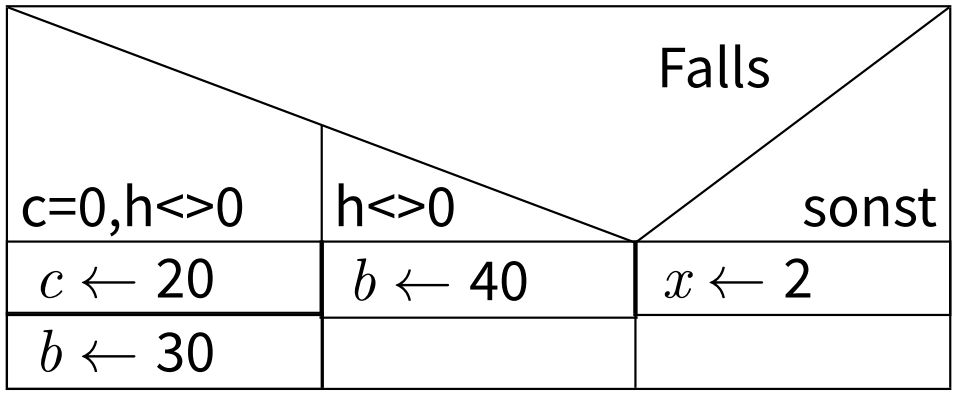


Bedingte Verzweigung

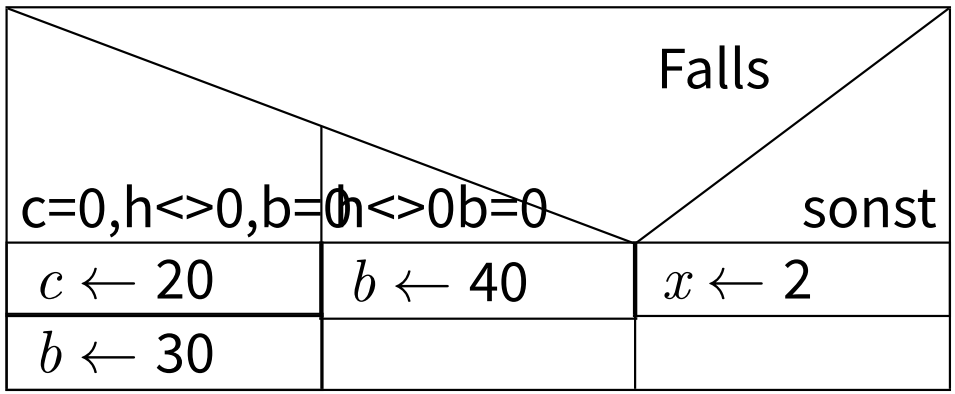


Elemente von Struktogrammen IV

Bedingte Mehrfachverzweigung



Fallunterscheidung



Elemente von Struktogrammen V

Abbruchanweisung

Aufgaben zum Knobeln

Terminieren die folgenden Algorithmen:

- Nehmen Sie ein Atom nach dem anderen (ohne eines doppelt zu betrachten) und zählen sie diese.
- Sie bekommen eine Liste von endlich vielen Zahlen in beliebiger Reihenfolge.
  1. Testen Sie, ob die Zahlen sortiert sind, wenn sie sortiert sind, sind sie fertig.
  2. Vertauschen Sie zwei beliebige Zahlen.
  3. Wiederholen Sie die Schritte 1) und 2) bis Sie fertig sind.
- Sie sind in einem 2D Irrgarten. Sie durchlaufen den Garten nach folgendem Schema: Sie orientieren sich immer an ihrer rechten Wand und laufen so lange weiter, bis Sie den Ausgang erreicht haben.
- Ihr Banknachbar denkt sich eine Ganzzahl aus. Sie starten bei 1 und erhöhen jeweils um 1 bis Sie die gesuchte Zahl erraten haben.