

Einführung in die Informatik

für Maschinenbau, Wirtschaftsingenieurwesen und
Verfahrenstechnik

Vorlesung

Ao.Univ.Prof. Dr. Gerald Futschek



Institut für Softwaretechnik
und Interaktive Systeme

Kapitel 1: Einführung

Inhalte Kapitel 1

- Was ist Informatik?
- Zentraler Begriff „Algorithmus“
- Eigenschaften von Algorithmen
- Notation von Algorithmen
- Syntax und Semantik



Was ist Informatik?

- **Theoretische Informatik**

(Schnittstelle zur **Mathematik**)

- Algorithmentheorie, Komplexitätstheorie, Berechenbarkeit, Formale Sprachen, Automatentheorie, Kodierungstheorie, Logik, ...

- **Technische Informatik**

(Schnittstelle mit **Elektrotechnik**)

- Schaltungstechnik, Rechnerarchitektur, Betriebssysteme, Systemprogramme, Mikroprogrammierung, Robotik, Rechnernetze, ...

- **Praktische Informatik** (sog. **Kerninformatik**)

- Programmierung, Softwaretechnik, Informationssysteme, Computergrafik, Visualisierung, ...

- **Angewandte Informatik**

- Anwendung der Informatik in verschiedenen Disziplinen: medizinische Informatik, Bioinformatik, ...

- weiters: **Wirtschaftsinformatik**

Herkunft der Informatik

- **Mathematik (Formalwissenschaft)**

- Vorgabe von Axiomen und Ableiten von Aussagen.
 - Beispiel: $(x + y) + z = x + (y + z)$

- **Naturwissenschaften**

- Entwicklung formaler Modelle, um Beobachtungen der Natur zu erklären.
- Vorhersage neuer Beobachtungen durch Ableiten weiterer Aussagen.
 - Beispiel: Newtonsche Gesetze zur Bewegungslehre

- **Ingenieurwissenschaften**

- Konstruktion von technischen Systemen durch Ausnutzen von Modellen über die Natur; es entstehen Pläne zur konkreten Produktion.
 - Beispiel: Eisenbahnbrücke, Berechnungen zur Statik

Einordnung der Informatik

- Informatik ist eine Formalwissenschaft
 - Theoretische Grundlagen zu *Berechnungsverfahren*.
 - Modelle zur *Verarbeitung* und *Speicherung* von Informationen.
- Informatik ist eine Ingenieurwissenschaft
 - Stichwort „Software Engineering“.
 - Konstruktion von Programmsystemen
 - auf der Basis geeigneter formaler Modelle.
 - für den Ablauf auf konkreten Rechenanlagen (Computer).
 - Vorhersagbarkeit gewünschter Eigenschaften ist wichtig.

Vielfalt an Modellen

- Kontinuierliche Modelle (aus Mathematik und Physik)
 - Gleichungssysteme, partielle Differentialgleichungen
 - statistische und wahrscheinlichkeitstheoretische Beschreibungen
 - ...
- Diskrete Modelle (aus diskreter Mathematik und Informatik)
 - Entscheidungstabellen
 - Graphstrukturen: gerichtete und ungerichtete Graphen, Bäume
 - Zustandssysteme: endliche Automaten, Petrinetze
 - formale Sprachen: Grammatiken, Syntaxdiagramme
 - ...

Zentraler Begriff *Algorithmus*

Ein **Algorithmus** ist ein *schrittweises, präzises Verfahren* zur Lösung eines Problems.

benannt nach Al Chwarismi, Persischer Mathematiker, 780 - 846

Beispiel für Algorithmus

- Aufgabe: Berechne die Summe der Zahlen von 1 bis n .
- Algorithmus *SummeBis*($\downarrow n$):
 - setze** *summe* $\leftarrow 0$
 - setze** *zähler* $\leftarrow 1$
 - solange** *zähler* $\leq n$:
 - setze** *summe* \leftarrow *summe* + *zähler*
 - erhöhe** *zähler* **um** 1
 - gib aus**: "Die Summe ist: " **und** *summe*
- Welches Ergebnis liefert *SummeBis*(n) für $n = 5$?

Bestandteile von Algorithmen

- Ein Algorithmus verarbeitet *Daten* mit Hilfe von *Anweisungen*.

Programm = Daten + Befehle

- Daten

- *Werte*: Zahlen (z. B. 0, 1, ...),
Strings ("Die Summe ist: "),
Zeichen ('a', 'b', ...)
- *Variablen*: Benannte Behälter für Werte (*summe*, *zähler*, *n*)

- Anweisungen

- Zuweisung: **setze** *n* \leftarrow 1
- bedingte Anweisung: **falls** \langle Bedingung \rangle : \langle Anweisung \rangle
- Folgen von Anweisungen: \langle Anweisung 1 \rangle ... \langle Anweisung *k* \rangle
- Schleifen: **solange** \langle Bedingung \rangle : \langle Anweisung \rangle

Basiseigenschaften von Algorithmen

- Allgemeinheit
 - Lösung einer Klasse von Problemen, nicht eines Einzelproblems.
- Operationalität
 - Einzelschritte sind wohldefiniert und können auf entsprechend geeigneten Rechenanlagen **ausgeführt** werden.
- Endliche Beschreibung
 - Die Notation des Algorithmus hat eine endliche Länge.
- Funktionalität
 - Ein Algorithmus reagiert auf Eingaben und produziert Ausgaben.

Weitere Eigenschaften von Algorithmen

- Terminierung Alg. läuft für jede Eingabe nur endlich lange
- Vollständigkeit Alg. liefert alle gewünschten Ergebnisse
- Korrektheit Alg. liefert nur richtige Ergebnisse
- Determinismus Ablauf ist für dieselbe Eingabe immer gleich
- Determiniertheit Ergebnis ist eindeutig festgelegt für jede Eingabe
- Effizienz Alg. ist sparsam im Ressourcenverbrauch
- Robustheit Alg. reagiert sinnvoll bei Fehlern und unerwarteten Eingaben
- Änderbarkeit Alg. ist anpassbar an modifizierte Anforderungen

Algorithmen: Terminierung

- Ein **terminierender** Algorithmus läuft für jede (!) beliebige Eingabe jeweils in endlicher Zeit ab.
- Gegenbeispiel:
 - Berechnung des Produktes der ersten n natürlichen Zahlen
 - $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1 = n \cdot (n-1)!$

Algorithmus:

$\text{fak}(\downarrow n)$: falls $n = 0$, liefere 1
sonst liefere $n \cdot \text{fak}(n-1)$

- Ergebnis von $\text{fak}(2)$?
- Ergebnis von $\text{fak}(4)$?
- Ergebnis von $\text{fak}(-3)$?

Algorithmen: Vollständigkeit

- Ein **vollständiger** Algorithmus gibt alle gewünschten Ergebnisse aus.
- Gegenbeispiel
 - Teilermenge ($\downarrow n$):
setze $i \leftarrow 1$
solange $i < \sqrt{n}$:
 falls n/i ganzzahlig:
 dann gib i und n/i aus
 erhöhe i um 1
 - Ausgabe von Teilermenge (12)?
 - Ausgabe von Teilermenge (9)?

Algorithmen: Korrektheit

- Ein **korrekter** Algorithmus liefert nur richtige Ergebnisse.
- Gegenbeispiel
 - Teilermenge2 ($\downarrow n$):
setze $i \leftarrow 1$
solange $i \leq \sqrt{n}$:
 gib i und n/i aus
 erhöhe i um 1
 - Ausgabe von Teilermenge2 (12)?
 - Ausgabe von Teilermenge2 (9)?

Algorithmen: Determinismus

- Ein **deterministischer** Algorithmus läuft für ein- und dieselbe Eingabe immer auf dieselbe Art und Weise ab.
- Ein **nichtdeterministischer** Algorithmus kann für ein- und dieselbe Eingabe unterschiedlich ablaufen.
- Beispiel:
 - Absolutbetrag(x):
wähle gültigen Fall $\left\{ \begin{array}{l} \text{falls } x \leq 0, \text{ liefere } -x \\ \text{falls } x \geq 0, \text{ liefere } x \end{array} \right.$
 - Ergebnis von Absolutbetrag (3)?
 - Ergebnis von Absolutbetrag (-5)?
 - Ergebnis von Absolutbetrag (0)?

Algorithmen: Determiniertheit

- Ein **determinierter** Algorithmus liefert für ein- und dieselbe Eingabe immer dasselbe Ergebnis.
- Gegenbeispiel
 - RotGrünBlau ($\downarrow n$):
wähle Fall: $\left\{ \begin{array}{l} \text{falls } n/2 \text{ ganzzahlig:} \\ \text{falls } n/3 \text{ ganzzahlig:} \\ \text{sonst:} \end{array} \right. \begin{array}{l} \text{liefere „rot“} \\ \text{liefere „grün“} \\ \text{liefere „blau“} \end{array}$
 - Ergebnis von RotGrünBlau (5)?
 - Ergebnis von RotGrünBlau (6)?
 - Wichtiges nicht-determiniertes Beispiel: Zufallszahlengenerator

Algorithmen: Effizienz

- Für eine gegebene Eingabe sollen die benötigten Ressourcen möglichst gering (oder sogar minimal) sein.
 - Betrachtung von *Speicherplatz* und *Rechenzeit* (Anzahl Einzelschritte).
 - Ggf. auch Analyse von *Plattenzugriffen* (I/Os) oder *Netzzugriffen*.
- Beispiel
 - Die iterative Berechnung der Summe von 1 bis n benötigt n Schritte. Zeitaufwand wächst proportional zu n , ist daher **linear** ($O(n)$, Ordnung n)
 - Verwende Summenformel $n \cdot (n+1)/2$ für einen **effizienteren** Algorithmus! Zeitaufwand ist **konstant** ($O(1)$, Ordnung 1)
- Unterscheide Effizienz und Effektivität
 - **Effektivität** ist „Grad der Zielerreichung“ (Vollständigkeit, Korrektheit).
 - **Effizienz** ist „Wirtschaftlichkeit der Zielerreichung“.

Beispiel: Suche in einem Array

- Aufgabe

Sei $A[1] \dots A[n]$ eine Reihe („Array“) von n verschiedenen Zahlen.
Suche jene Position i , bei der eine gegebenen Zahl x vorkommt, also das i mit $A[i] = x$.
- Algorithmus **SequentielleSuche** (A, x):

Durchlaufe das Array A der Reihe nach für $i \leftarrow 1, \dots, n$:
Falls $A[i] = x$: gib i aus und beende den Durchlauf.
Falls x nicht gefunden, gib Fehlermeldung „nicht gefunden“ aus.
- Anzahl der Vergleiche (= Analyse der Laufzeit)
 - Erfolgreiche Suche: n Vergleiche maximal, $n/2$ im Durchschnitt
 - Erfolgreiche Suche: n Vergleiche
 - Laufzeit wächst **linear** mit n : $O(n)$ (sprich: Ordnung n)

Beispiel: Binäre Suche

Falls das Array **sortiert** ist, kann man x in A auch wie folgt suchen:

BinäreSuche (A, x)

Vergleiche den mittleren Eintrag $A[\text{mitte}]$ mit x :

Falls $x = A[\text{mitte}]$, gib mitte aus und beende die Suche.

Falls $x < A[\text{mitte}]$, suche in der linken Hälfte von A weiter.

Falls $x > A[\text{mitte}]$, suche in der rechten Hälfte von A weiter.

In der jeweiligen Hälfte wird ebenfalls mit *BinäreSuche* gesucht.

Falls die neue Hälfte des Arrays leer ist, gib die Fehlermeldung „nicht gefunden“ aus.

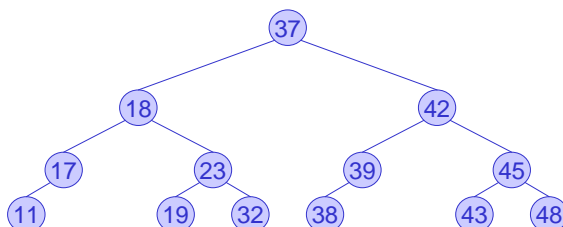
Zeitaufwand ist proportional zu $\log_2 n$ (*logarithmische* Ordnung, $O(\log n)$)

Beispiel für die binäre Suche

- Beispiel

11	17	18	19	23	32	37	38	39	42	43	45	48
----	----	----	----	----	----	----	----	----	----	----	----	----

- Entscheidungsbaum



- Analyse der Laufzeit

- Entscheidungsbaum hat Höhe $h = \lceil \log_2(n) \rceil$.
- Suche benötigt maximal h viele Vergleiche.

- Vergleich der Suchverfahren

- Beispiel $n = 1.000$
 - Sequentiell: 1.000 Vergleiche
 - Binäre Suche: 10 Vergleiche
- Beispiel $n = 1.000.000$
 - Sequ.: 1.000.000 Vergleiche
 - Binäre Suche: 20 Vergleiche

Algorithmen: Robustheit

- Der Ablauf soll auch für fehlerhafte Eingaben oder Daten wohldefiniert sein.
- Beispiele
 - **Bedienfehler**
 - Leere Eingaben in (grafische) Eingabefelder
 - Eingabe von Strings in Eingabefelder für Zahlen
 - **Systemfehler**
 - Zugriff auf Ressourcen (Dateien, Netz, etc.) nicht möglich

Algorithmen: Änderbarkeit

- Einfache Anpassung an veränderte Aufgabenstellungen.
- Beispiele
 - Erweiterung einer Adressenverwaltung für inländische Kunden auf Kunden im Ausland
 - Umstellung der PLZ von vier auf fünf Stellen
 - Einführung des EURO
 - „Jahr 2000“ Problem (vierstellige statt zweistellige Jahreszahlen)

Programme und Prozesse

- Programm
 - Beschreibung eines Ablaufes, der auf einer Rechenanlage durchgeführt werden kann (= Algorithmus, *statisch*).
 - Programme sind (meist) in Dateien gespeichert.
 - Programme entstehen durch Konstruktion.
- Prozess
 - Konkreter Ablauf eines Programms (*dynamisch*).
 - Prozesse benötigen Betriebsmittel (Rechenzeit, Speicher, Dateizugriffe).
 - Prozesse werden durch das Betriebssystem verwaltet.
 - Prozesse entstehen durch Aufrufe von Programmen.

Übersetzung von Programmen

- Problemstellung
 - Programme werden meist in höheren Programmiersprachen formuliert.
 - Die Hardware kann nur Programme in Maschinsprache ausführen.
 - Abbildung höherer Programmiersprachen in Maschinsprachen nötig.
 - Übersetzung notwendig
- Zwei Vorgehensweisen
 - **Compilation:**
Programme werden vor dem Ablauf übersetzt (compiliert).
 - **Interpretation:**
Programme werden während des Ablaufs übersetzt (interpretiert).

Compilation und Interpretation

- **Compilation**

- Das Programm wird nur einmal analysiert (*statisch*).
- Das resultierende (Maschinen-)Programm wird gespeichert.
- Dieser Übersetzer heißt **Compiler**.
- Die Ausführung ist in der Regel schneller als bei Interpretation.

- **Interpretation**

- Das Programm wird bei jeder Ausführung analysiert (*dynamisch*).
- Dieser Übersetzer heißt **Interpreter**.
- Die Ausführung ist in der Regel langsamer als bei Übersetzung.
- Die Zyklen „Programmänderung → Ausführung“ sind kürzer.

Vergleich: Übersetzer - Simultandolmetscher

Übersetzungs- und Interpretationshierarchie

- Die Abbildung auf maschinennähere Sprachen kann sich über mehrere Stufen erstrecken.
- Beispiel Java:



Hardware und Software

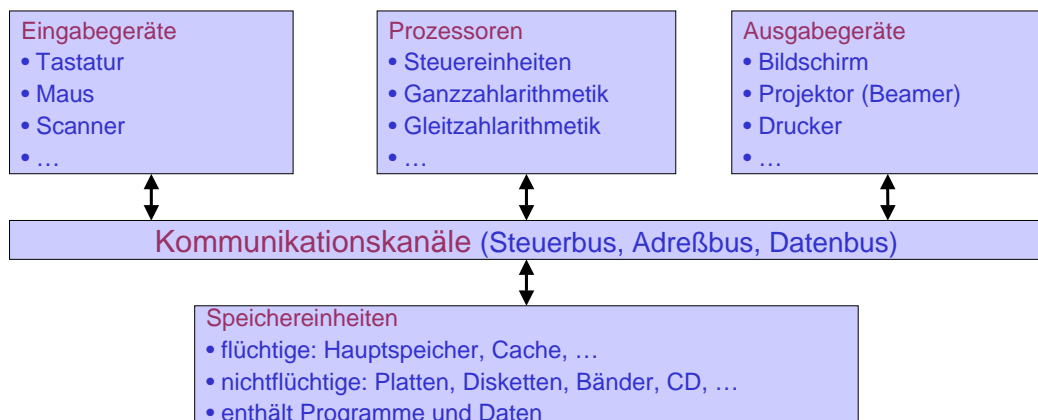
- Hardware
 - „Harte Ware“: Geräte
 - Prozessor, Speicher (flüchtig/nicht-flüchtig), Ein-/Ausgabegeräte
 - Modellbeispiel: „von Neumann-Rechner“
- Software
 - „Weiche Ware“: Programme, Informationen, Daten
 - Systemprogramme zur Steuerung der Hardware
 - Anwendungsprogramme zur Bearbeitung von Nutzeraufgaben
 - Programmdokumentation
- Firmware
 - In Hardware gegossene Software (Gerätesteueringen, Mikroprogramme)

Rechnermodell von John von Neumann



John von Neumann

1903 - 1957



Notationen für Algorithmen

- Informell
 - Natürliche Sprache
 - Pseudocode (halbformell)
- Graphische Darstellungen
 - Programmablaufplan (Flussdiagramm, DIN 66001)
 - Struktogramm (Nassi-Shneiderman-Diagramm, DIN 66261)
- Formale Sprachen (Programmiersprachen)
 - Programm als formaler Text

Beispielalgorithmus *SummeBis*(n)

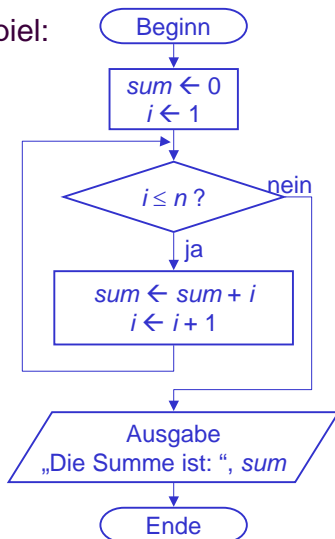
- Aufgabe: Berechne die Summe der Zahlen von 1 bis n .
- Natürliche Sprache

Initialisiere eine Variable *summe* mit 0. Durchlaufe die Zahlen von 1 bis n mit einer Variable *zähler* und addiere *zähler* jeweils zu *summe*. Gib nach dem Durchlauf den Text "Die Summe ist: " und den Wert von *summe* aus.
- Pseudocode

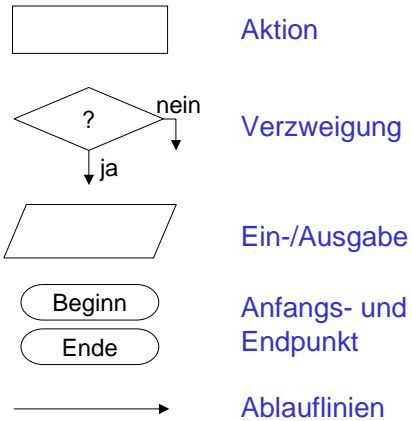
```
setze summe ← 0
setze zähler ← 1
solange zähler ≤  $n$ :
    setze summe ← summe + zähler
    erhöhe zähler um 1
gib aus: "Die Summe ist: " und summe
```

Flußdiagramm für *SummeBis(n)*

Beispiel:

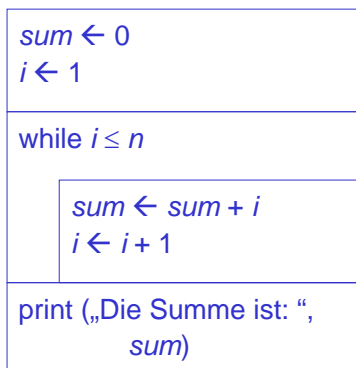


Symbole:

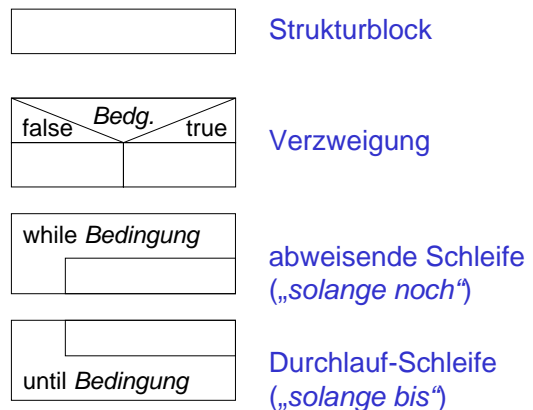


Struktogramm für *SummeBis(n)*

Beispiel:



Symbole:



Beispiel SummeBis, Programmiersprache Java

```
class SummeBis {  
    public static int sumbis (int n) {  
        int sum = 0;  
        int i = 1;  
        while (i <= n) {  
            sum = sum + i;  
            i = i + 1;  
        }  
        return sum;  
    }  
}  
  
class Test_of_SummeBis {  
    public static void main (String[] arg) {  
        int k = Integer.parseInt(arg[0]);  
        System.out.println ("Die Summe ist: " + SummeBis.sumbis(k));  
    }  
}
```

„Schreibtischtest“ für Algorithmen

- Veranschaulichung der Funktionsweise eines Algorithmus durch ein *Ablaufprotokoll* der Variableninhalte.

Beispiel: Vertausche die Inhalte der Variablen x und y

Algorithmus **tausche** ($\Downarrow x$, $\Downarrow y$)

1. Versuch:

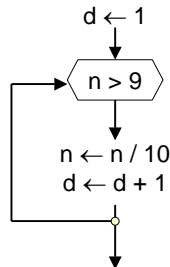
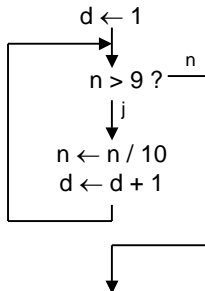
	x	y
$x \leftarrow y$	3	5
$y \leftarrow x$	5	5

2. Versuch: mit Hilfsvariable

	x	y	h
$h \leftarrow x$	3	5	
$x \leftarrow y$	5		3
$y \leftarrow h$		3	

Anzahl der Dezimalziffern einer Zahl

Algorithmus *NumDigits*($\downarrow n, \uparrow d$)



„Schreibtischtest“

n	d
437	1
43	2
4	3

1. Aufgabe: Algorithmus für x^n

- **Gegeben:** reelles x , ganzzahliges n
- **Gesucht:** ein Algorithmus zur Berechnung von x^n
Beachten Sie, dass sowohl x als auch n auch negativ oder gleich Null sein können.
- **Beispiele:**
 $3^4 = 81$, $(-2)^5 = -32$, $3.45^0 = 1$, $(-2.5)^2 = 6.25$, $2^{-3} = 1/(2^3) = 0.125$
- Lösen Sie die Aufgabe bis zur nächsten Vorlesung.
- Formulieren Sie den Algorithmus im Pseudocode oder als Ablaufdiagramm

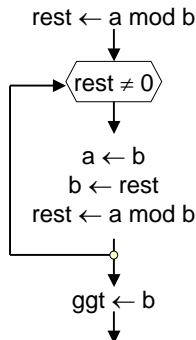
Korrektheit: Euklidischer Algorithmus

- Aufgabe: größter gemeinsamer Teiler zweier Zahlen
- Lösung von Euklid



Euklid, um 300 v. Chr.

- Euklid ($\downarrow a, \downarrow b, \uparrow ggt$):



Nachweis der Korrektheit

$$(\text{ggT teilt } a) \wedge (\text{ggT teilt } b)$$

→ ggt teilt $(a - b)$

→ ggt teilt $(a - k^*b)$

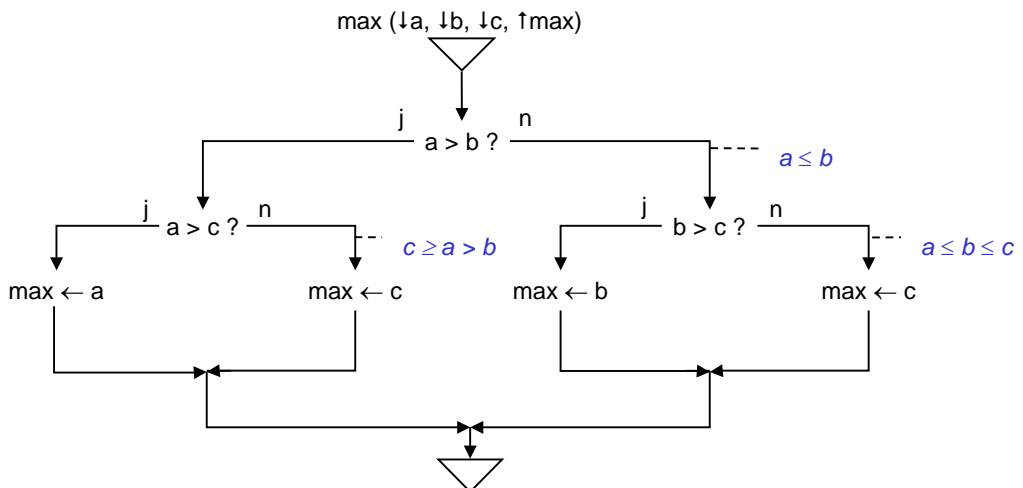
→ ggt teilt $rest$

→ $\text{ggt}(a, b) = \text{ggt}(b, \text{rest})$

Beobachtung: Wert von $ggT(a,b)$ bleibt unverändert!

Verwendung von Zusicherungen

Beispiel: Maximum dreier Zahlen a, b, c

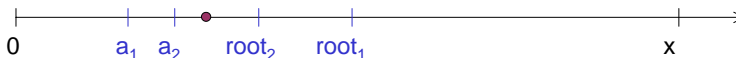


2. Aufgabe: Triple-Sort

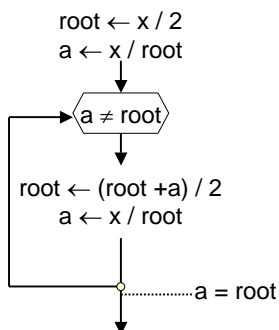
- Zu lösen bis zur nächsten Vorlesung:
- Gegeben sind drei ganze Zahlen a, b, c
- Entwerfen Sie einen Algorithmus, der die 3 Zahlen aufsteigend sortiert.
- Ganz egal welche Werte die 3 Zahlen vor Beginn des Algorithmus hatten, gilt nach dem Algorithmus $a \leq b \leq c$.
- Formulieren Sie den Algorithmus im Pseudocode oder als Ablaufdiagramm

Nicht-ganzzahlige Arithmetik

- Bsp: Berechnung der Quadratwurzel durch schrittweise Näherung



Quadratwurzel ($\downarrow x, \uparrow \text{root}$):



x	root	a
10	5	2
	3.5	2.85714
	3.17857	3.14607
	3.16232	3.16223
	3.16228	3.16229
	3.16229	3.16228

Problem mit Terminierung:

$a = \text{root}$ wird (fast) nie erreicht,
besser: $|a - \text{root}| > 1.0\text{e-}8$ testen.

Formale Sprachen

- Syntax
 - Regeln, nach denen Programme aufgeschrieben werden dürfen.
 - Beispiel: *Zuweisung* = *Variable* " \leftarrow " *Ausdruck*.

Welche der folgenden Zuweisungen sind syntaktisch korrekt?

- $A \leftarrow 3 + 5$
 - $B \leftarrow \leftarrow 2$
 - $7 \leftarrow 4$
- Semantik
 - Bedeutung von korrekt gebildeten Programmen.
 - Im Beispiel: *Weise den Wert des Ausdrucks an die Variable zu*
 - Nach „ $A \leftarrow 3 + 5$ “ enthält Variable A den Wert der Summe 3+5, also 8.

EBNF: Erweiterte Backus-Naur-Form

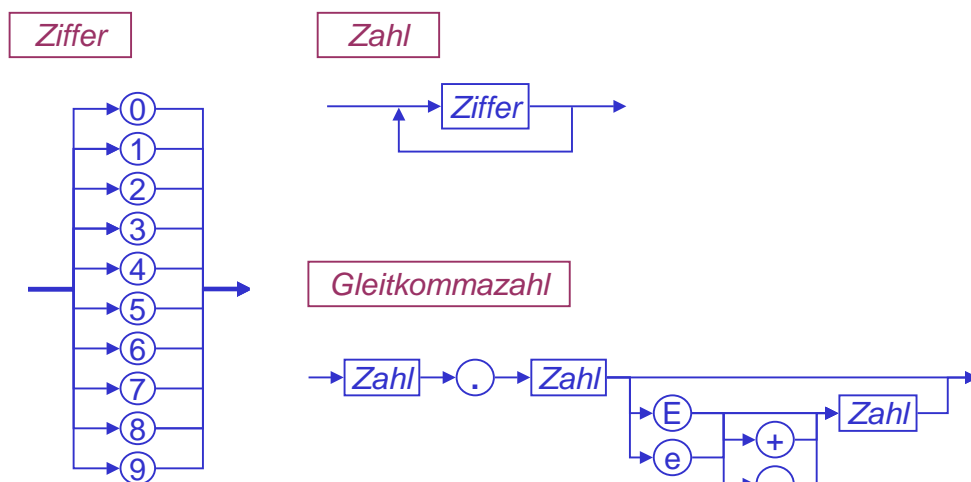
- EBNF: einfache textuelle Darstellung von Grammatiken
 - John Backus: Entwickler der Programmiersprache *Fortran*
 - Peter Naur: Herausgeber des Algol60-Reports

Metazeichen für EBNF-Regeln	Bsp.	steht für
=		trennt Regelseiten
.		schließt Regel ab
	$x \mid y$	trennt Alternativen
()	$(x \mid y) z$	klammert Alternativen
[]	$[x] y$	wahlweises Vorkommen
{ }	$\{x\} y$	0..n-maliges Vorkommen
		x, y
		xz, yz
		xy, y
		y, xy, xxy, \dots

Beispiel: Grammatiken für Zahlen

- Beispiel: Grammatik für ganze Zahlen (ohne Vorzeichen: 1, 4711, ...)
 $Ziffer = "0" \mid "1" \mid "2" \mid "3" \mid "4" \mid "5" \mid "6" \mid "7" \mid "8" \mid "9".$
 $Zahl = Ziffer \{ Ziffer \}.$
- Aufbau von Regeln für Grammatiken
 - *Terminalsymbole* werden nicht mehr weiter zerlegt (im Bsp. in " ").
 - *Nichtterminalsymbole* werden ersetzt (im Bsp. kursiv).
 - Auf der linken Seite einer Regel steht immer ein Nichtterminalsymbol.
 - Rechte Seite kann Terminalsymbole und Nichtterminalsymbole enthalten.
- Beispiel: Gleitkommazahlen (ohne Vorzeichen: 1.0E-2, 3.14159E0, ...)
 $Gleitkommazahl = Zahl "." Zahl [("E" \mid "e") ["+" \mid "-"] Zahl].$

Grafische Alternative: Syntaxdiagramme



3. Aufgabe: Syntax der Österr. Autokennzeichen

- Die Österreichischen Autonummerntafeln haben (vereinfacht) zunächst ein 1 oder 2 Buchstaben für den pol.Bezirk, danach entweder zuerst mehrere Buchstaben, ein Leerfeld und dann Ziffern, oder umgekehrt zuerst Ziffern und dann Buchstaben.
- Schreiben Sie die Syntax der Autokennzeichen
 - a) in Backus-Naur Form
 - b) in Form von Syntaxdiagrammen
- Zu lösen bitte bis zur nächsten Vorlesung

Programmierung und Softwareentwicklung

- „Programmierung im Kleinen“
 - Konstruktion von Teilprogrammen
 - Anweisungen, Prozeduren, Objekte, Klassen
 - Zusammenfassung in Bibliotheken
- „Programmierung im Großen“
 - „Software Engineering“
 - Konstruktion großer Programmsysteme mit Hilfe von Bibliotheken
 - Vorgehensmodelle zur Zerlegung und Integration

Paradigmen der Programmierung

- **Imperative Programmierung**
 - Folgen von Anweisungen, Verzweigungen, Schleifen, Prozeduraufrufe
 - Beispiele: BASIC, Pascal, Modula2, ...
- **Objektorientierte Programmierung**
 - Klassen beschreiben Struktur und Verhalten interagierender Objekte
 - Beispiele: Simula, Smalltalk, C++, Java, ...
- **Logik- oder deklarative Programmierung**
 - Automatische Ableitung der Lösung aus der Beschreibung der Aufgabe
 - Beispiele: Prolog, ...
- **Funktionale oder applikative Programmierung**
 - Rekursive Anwendung von Funktionen
 - Beispiele: LISP, Scheme, ...