Grundlagen der Programmierung

Vom Problem zum Algorithmus:

Algorithmisches Denken ♦ Pseudocode

Algorithmisches Denken: Vom Problem zur Lösung

- *Problemstellung* in einem Bereich der Realität
- Wunsch nach *automatisierter Lösung*
- **Voraussetzung:** *Algorithmus*

Beispiele

- **größter gemeinsamer Teiler (ggT):**
  Welche Zahl ist ggT von zwei natürlichen Zahlen?

- **größtes Listenelement:**
  Welches ist die größte Zahl in einer Liste ganzer Zahlen?

- **Freundschaftsproblem:**
  Wie oft kommt es vor, dass sich Freundschaften als transitive Beziehung erweisen?

1. Identifizieren des Problems

- Ausgangspunkt: Problem in einer Anwendungsdomäne
- zwei Rollen (*logische Sicht*):
  - **Domänenexperte**
    - verfügt über Daten (**Eingabedaten**)
    - stellt Frage, deren Antwort nicht direkt in den Daten abzulesen ist
  - **Entwickler**
    - soll Antwort aus den Daten gewinnen (**Ausgabedaten**)
    - auf einheitliche Weise (unabhängig von den konkreten Eingabedaten)
- Identifizieren des Problems = Identifizieren der I/O-Daten

Erinnerung: Algorithmus (intuitiv)

- Ein **Algorithmus** ist eine Folge von Anweisungen, die **Eingabedaten** in **Ausgabedaten** überführt.
- Dabei muss für jede Eingabe eindeutig feststehen:
  - Welche Anweisung wird zuerst ausgeführt?
  - Welche Anweisung folgt auf eine gerade ausgeführte?
  - Wann ist der Algorithmus beendet?
- Der Algorithmus muss für alle (passenden) Eingabedaten die Ausgabedaten korrekt berechnen, ohne dass er angepasst werden muss.

2. Formulieren des Problems

- In welcher Form können die Eingabe- und Ausgabedaten repräsentiert werden?

ggT
Eingabe: zwei natürliche Zahlen
Ausgabe: eine natürliche Zahl

größtes Listenelement
Eingabe: Liste ganzer Zahlen
Ausgabe: eine ganze Zahl

→ *Wie greift man auf Listenelemente zu? Wie bestimmt man die Länge der Liste? ...*

Freundschaftsproblem
???

3. Entwurf des Algorithmus

- **Wie** werden die Eingabedaten in die Ausgabedaten überführt? (*Folge von Anweisungen*)

- **Korrekt?** (für alle Eingabedaten)
- **Terminiert?** (für alle Eingabedaten)
- **Effizient?** (für alle Eingabedaten)

- Donald Knuth: *The Art of Computer Programming.* Addison-Wesley, 1997/2005.

4. Implementieren des Algorithmus

- Formulierung der algorithmischen Idee
  in einer Sprache, die auf einer Maschine
  ausgeführt werden kann (**Programmiersprache**).

  → **Programmieren**

- Erhaltung der Eigenschaften von 1., 2. und 3.!!!

5. Anwenden des Algorithmus

- Ausführen des Programms auf einer Maschine (mit konkreten Eingabedaten)
  → *Beantwortung der Fragestellung*

- korrekte Berechnungen ← erfolgreiches **Testen**

  !! *liefert keinen Beweis für Korrektheit des Algorithmus!*

Algorithmisches Denken: Vom Problem zur Lösung

1. **Identifizieren des Problems**
2. **Formulieren des Problems**
3. **Entwurf des Algorithmus**
4. **Implementierung des Algorithmus**
5. **Anwendung des Algorithmus**

→ Problemlösung

Vom Problem zum Algorithmus

Beispiel 1

1. Spezifikation des Algorithmus:
   Name: **größtes Listenelement**
   Eingabe: Liste *L* ganzer Zahlen
   Ausgabe: größte Zahl in der Liste

2. Liste als **Folge** von Elementen (Zahlen), die durch die Nummer ihrer Position (**Index**) aufgefunden werden: $L[1], L[2], L[3], \ldots$
   → **indizierte Liste**

   Beispiel
   L: 12, 3, 7, 8, 1

Index i	$L[i]$
1	12
2	3
3	7
4	8
5	1

→ **Länge** der Liste: $|L|$

Beispiel 1

3. Algorithmische Idee:

   Durchlaufe alle Listenelemente und merke immer die Zahl, die bisher die größte war.

- Für spätere Implementierung benötigen wir eine präzisere Sprache: **Pseudocode** (*semi-formale Sprache*)
  - „zwischen“ natürlicher und Programmiersprache, spezialisiert zum Beschreiben von Algorithmen
  - möglichst keine Mehrdeutigkeiten
  - kann/darf nicht vollständig formalisiert werden

Beispiel 1 - Pseudocode

Name: **größtes Listenelement**
Eingabe: Liste L ganzer Zahlen
Ausgabe: größte Zahl in der Liste

$x \longleftarrow$ größtes Element der Liste L
gib x aus

$x \longleftarrow -\infty$
für alle z in L
    falls $z > x$
        $x \longleftarrow z$
gib x aus

$x \longleftarrow -\infty$
für $i \longleftarrow 1$ bis $|L|$
    falls $L[i] > x$
        $x \longleftarrow L[i]$
gib x aus

Beispiel 1 - Pseudocode

Name: **größtes Listenelement**
Eingabe: Liste *L* ganzer Zahlen
Ausgabe: größte Zahl in der Liste

$x \longleftarrow$ größtes Element der Liste L
gib x aus

Verfeinerung: *schrittweise*

- mehr Details
- weniger Umgangssprache
- mehr *Wie*-Beschreibung

$x \longleftarrow - \infty$
für alle z in L
    falls $z > x$
        $x \longleftarrow z$
gib x aus

Verfeinerung

$x \longleftarrow - \infty$
für $i \longleftarrow 1$ bis $|L|$
    falls $L[i] > x$
        $x \longleftarrow L[i]$
gib x aus

Vorgehen algorithmische Idee

1. Vertraut machen mit der Problemdomäne (Fachwissen!!!)
2. verbale Formulierung des Algorithmus
3. Formulierung in Pseudocode
4. schrittweise **Verfeinerung** des Pseudocode
   - leichter zu implementieren
   - leichter zu analysieren (terminiert?, effizient?)

Beispiel 1 – alternativer Algorithmus

Name: größtes Listenelement
Eingabe: Liste *L* ganzer Zahlen
Ausgabe: größte Zahl in der Liste

sortiere *L* absteigend
gib *L*[1] aus

Verfeinerung?!

Algorithmische Konzepte: Variablen

- **Variablen**
  - haben einen **Namen** (*x*, *z*, *L*, *var*, *input*, ...)
  - dienen zum Merken von **Werten**
    - Eingabedaten
    - Berechnungsergebnisse, Zwischenergebnisse, Zähler, ...
  - Werte können durch Anweisungen verändert werden

- **Kollektionen**
  - sind spezielle Variablen
  - zum Merken einer Vielzahl von Werten
  - *typisches Beispiel:* **indizierte Listen:**
    Zugriff auf einzelne Werte über einen Index: *L*[*index*]

Algorithmische Konzepte: Bedingungen

- *Beispiele:* $z > x$, $L[i] > x$, $i \leq |L|$, L nicht leer, ...
- können Variablen enthalten
- werden in Abhängigkeit vom Wert der Variablen wahr oder falsch (Aussageformen/Prädikate)
- mehrere Bedingungen können zu einer Bedingung zusammengesetzt werden
  - aussagenlogische Operationen:
    UND, ODER, ENTWEDER-ODER, ...
  - *z.B.* $x > 0$ UND $x < y$
  - *z.B.* $i > |L|/2$ UND $i \leq |L|$

Arithmetische Ausdrücke

- setzen sich zusammen aus Variablen, Werten und Operationen auf Werten und Variablen, *z.B.*
  $n + 4$, $m - (3k + 1)$, $|L|/2$, $-\,var$
  → *Operationen wirken auf Teilausdrücke (Terme)*
  - *Term Operation Term* (bei zweistelligen Operationen)
  - *Operation Term* (bei einstelligen Operationen)
- haben einen Wert

Wichtige Operationen

- Entgegengesetztes (**-**)
- Addition (**+**), Subtraktion (**-**), Multiplikation (**·**)
- Division
  - exakte Division (**/**) — 5/2 = 2,5
  - ganzzahlige Division (**DIV**) — 5 DIV 2 = 2
  - Rest bei ganzzahliger Division (**MOD**) — 5 MOD 2 = 1
- **|*L*|** und ***L*[*i*]** sind Operationen (auf Liste *L*)
- ...

Algorithmische Konzepte: Anweisungen

- Anweisungen können im Prinzip beliebig formuliert werden, solange klar ist, was zu tun ist, *z.B.:*

  sortiere *L* absteigend

- *insbesondere für Listen L:*

  füge ... *L* hinzu *und* entferne ... aus *L*

- **Ausgabeanweisung**

  gib ... **aus**

- **STOP** (Algorithmus beenden)

Algorithmische Konzepte: Anweisungen

1. Zuweisung

- Zuweisung eines Wertes an eine Variable
- *Variable* ⟵ *Ausdruck*
    1. Berechnung des Werts des *Ausdrucks* (darf die *Variable* enthalten)
    2. Zuweisung dieses Werts an die *Variable*

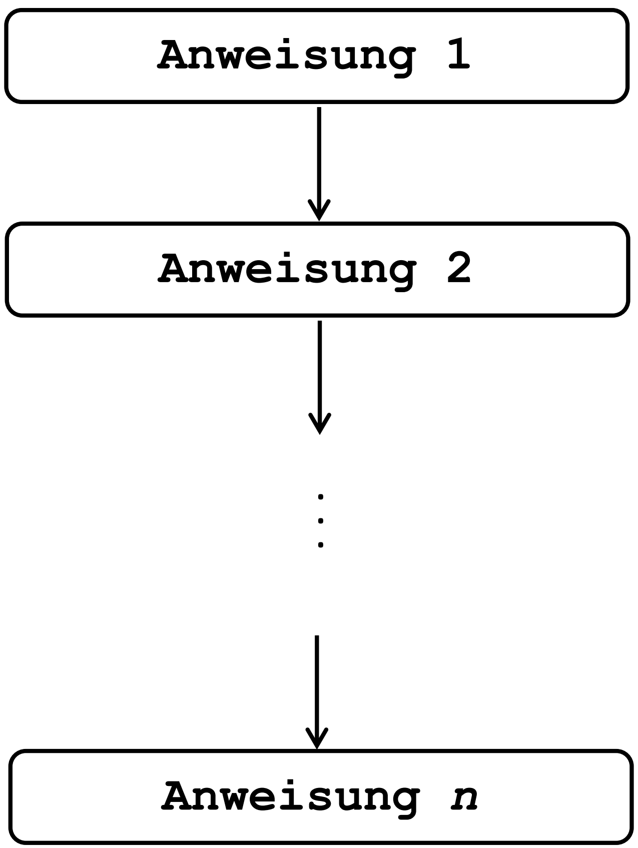
    Bsp: $x \leftarrow x + 1$

2. Sequenz

- Folge von Anweisungen; der Reihe nach abzuarbeiten
- *Beispiel:*

  sortiere *L* absteigend
  gib *L*[1] **aus**

Sequenz als Kontrollflussgraph (KFG)



bei gleicher Einrücktiefe: Anweisungs-block ($n \geq 1$)

Algorithmische Konzepte: Anweisungen

3. Fallunterscheidung

- **falls** *Bedingung*
  Anweisungsblock

  (***bedingte Anweisung***)

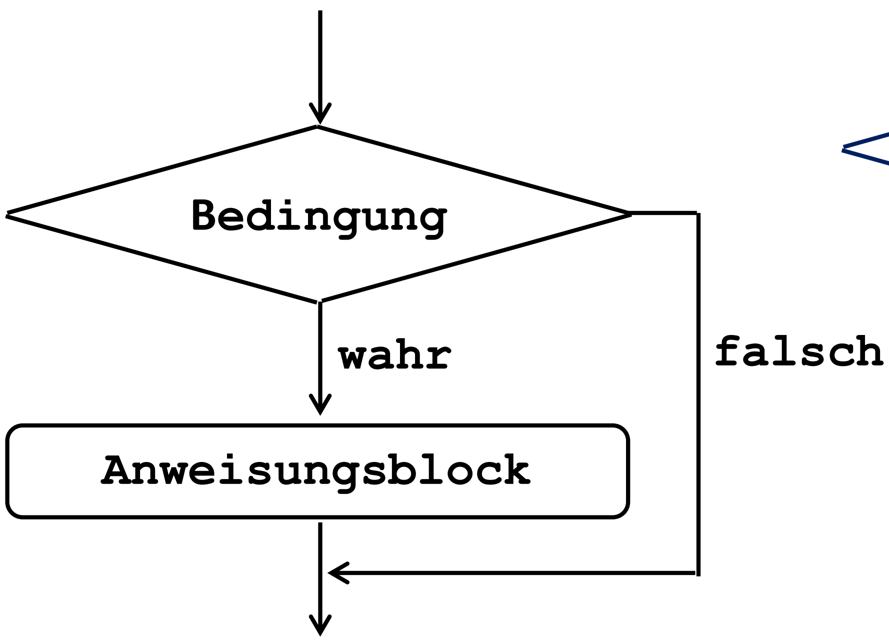
- **falls** *Bedingung*
  Anweisungsblock
  sonst
  Anweisungsblock

  (***alternative Anweisung***)

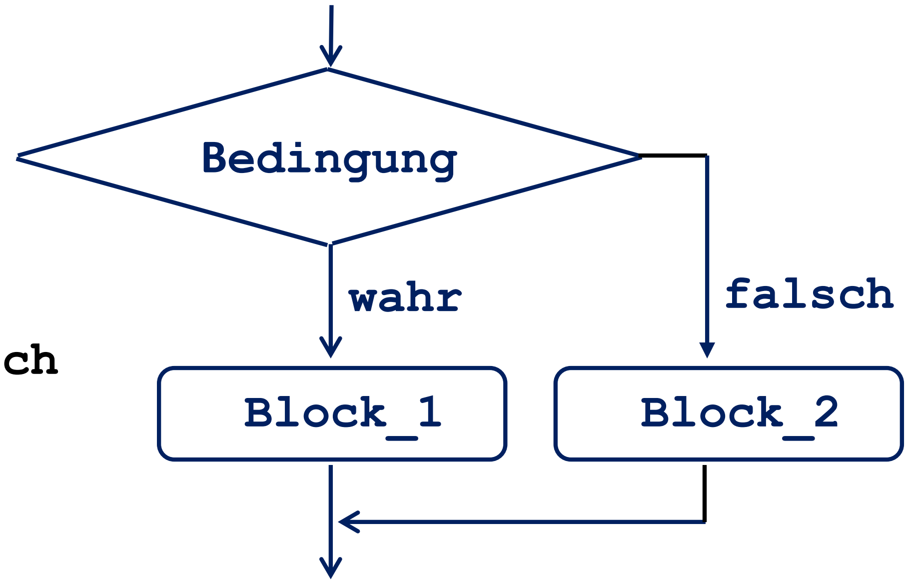
- ohne **sonst:** weiter mit nächster Anweisung
- *Beispiel:*
  falls $z > x$
  $x \longleftarrow z$
  gib x **aus**
- eingerückte Anweisung gehört zu **falls** bzw. **sonst**

Fallunterscheidung als KFG

Bedingte Anweisung



Alternative Anweisung



Algorithmische Konzepte: Anweisungen

4. Wiederholung

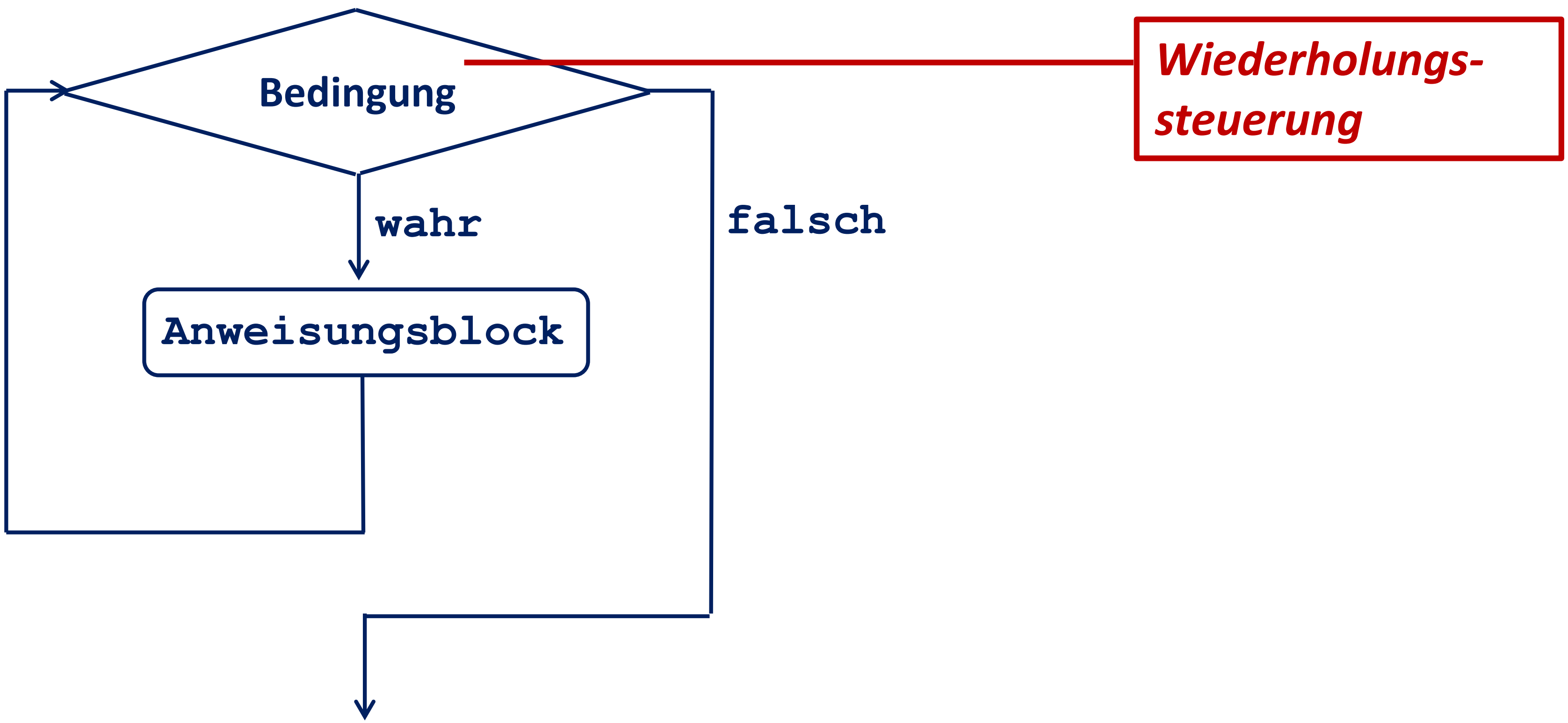
- *Beispiele:*
  für alle z **in** L
  Anweisungsblock

  für $i \longleftarrow 1$ **bis** $|L|$
  Anweisungsblock

- allgemein:
  Wiederholungssteuerung
  Anweisungsblock

- Die Anweisungen im *Anweisungsblock* werden wiederholt ausgeführt, solange es die *Wiederholungssteuerung* verlangt; dann weiter mit nächster Anweisung

Wiederholung als KFG



Arten der Wiederholungsteuerung

- **für alle** *Variable* **in** *Kollektion*

- **für** *Variable* $\longleftarrow$ *Ausdruck* **bis** *Ausdruck*
  - dabei Variablenwert schrittweise um 1 erhöhen

- **solange** *Bedingung*

  $i \longleftarrow 1$
  solange $i \leq |L|$ $\longleftrightarrow$ **für** $i \longleftarrow 1$ **bis** $|L|$

- **führe aus** *Anweisungsblock* **bis** *Bedingung*

Algorithmische Konzepte: Anweisungen

5. Fehlermeldung

- sorgt für aussagekräftige Fehlermeldung
- oft soll das Programm danach beendet werden
  → dann folgt die **STOP**-Anweisung
- **zeige** „ ... *Fehlermeldung* ...“

- *Beispiel:*

  falls $y = 0$

  zeige „Fehler: Divisor ist 0.“

  STOP

  $z \longleftarrow x / y$

  gib z **aus**

Algorithmus – erste Präzisierung

- Ein Algorithmus ist eine **Sequenz** (Anweisungsfolge), die Eingabedaten in Ausgabedaten überführt.

- Der Algorithmus ist beendet (**terminiert**), wenn es keine nächste Anweisung gibt oder eine STOP-Anweisung erreicht wurde.

Beispiel 2

Name: **größter gemeinsamer Teiler (ggT)**
Eingabe: zwei positive ganze Zahlen *x* und *y*
Ausgabe: ggT von *x* und *y*

Idee:

1. Bestimme alle Teiler von *x*
2. Bestimme alle Teiler von *y*
3. Suche den größten Teiler von *x*, der auch Teiler von *y* ist

Beispiel 2

Name: **größter gemeinsamer Teiler (ggT)**
Eingabe: zwei positive ganze Zahlen x und y
Ausgabe: ggT von x und y

$L_1 \longleftarrow$ Liste aller Teiler von x (aufsteigend sortiert)
$L_2 \longleftarrow$ Liste aller Teiler von y (aufsteigend sortiert)
$i \longleftarrow |L_1|$
solange $i \geq 1$
    falls $L_1[i]$ ist in L_2
        gib $L_1[i]$ **aus**
        STOP
    $i \longleftarrow i - 1$
gib 1 **aus**

Beispiel 2 (für die Verfeinerung)

Name: **Liste aller Teiler**
Eingabe: eine positive ganze Zahl x
Ausgabe: Liste mit allen Teilern von x

$i \longleftarrow 1$
für $k \longleftarrow 1$ **bis** x
    falls x MOD $k = 0$
        $L[i] \longleftarrow k$
        $i \longleftarrow i + 1$
gib L **aus**

Beispiel 2 – eine Effizienzbetrachtung

Name: **größter gemeinsamer Teiler (ggT)**
Eingabe: zwei positive ganze Zahlen x und y
Ausgabe: ggT von x und y

falls $x > y$
    vertausche x und y
$L_1 \longleftarrow$ Liste aller Teiler von x (aufsteigend sortiert)
$L_2 \longleftarrow$ Liste aller Teiler von y (aufsteigend sortiert)
$i \longleftarrow |L_1|$
solange $i \geq 1$
    falls $L_1[i]$ ist in L_2
        gib $L_1[i]$ **aus**
        STOP
    $i \longleftarrow i - 1$
~~gib 1 aus~~

x = 48 → 1, 2, 3, 4, 6, 8, 12, 16, 24, 48
y = 8 → 1, 2, 4, 8

testen: 48, 24, 16, 12, 8
nach Tausch von x und y nur ein Test

Beispiel 2 – alternativer Algorithmus

Fachwissen:

- Euklidischer Algorithmus
  - *siehe Übungen*