

4.1 Ausdrücke

4.2 Funktionale Algorithmen

4.3 Anweisungen

4.4 Imperative Algorithmen

4.5 Funktionale und imperative Algorithmen in Java

4.3 Anweisungen

- Wir versuchen zunächst wieder eine informelle Annäherung

- Eine Funktionsvereinbarung

$$\textit{Strecke}(m,t,k) = (k*t*t) / (2*m)$$

kann intuitive gelesen werden als:

„Der Ergebniswert von Algorithmus *Strecke* für Eingabewerte m,t,k ist $(k*t*t) / (2*m)$ “

- Wir könnten den Ausdruck $(k*t*t) / (2*m)$ möglicherweise auch mit „Zwischenberechnungen“ durchführen, z.B. indem wir eine weitere Variable z vom Typ $\mathbb{R}$ einführen und vereinbaren:

$$z = 2*m; \quad \textit{Strecke}(m,t,k) = (k*t*t) / (z)$$

- Die Vereinbarung der Variable z kann als **Aktion** verstanden werden:
„Berechne das Zwischenergebnis $2*m$ und bezeichne diesen Wert mit z “
(dies entspricht offenbar einer Substitution $[z/2*m]$)
- Es ist (syntaktisch ausgedrückt durch „;“) explizit eine Reihenfolge zwischen dieser Aktion und der Berechnung des Gesamtergebnisses festgehalten

4.3 Anweisungen

- Diese Abfolge von Berechnungsschritten beschreibt einen Algorithmus in *imperativer (prozeduraler)* Form durch eine Menge von *Anweisungen*

Algorithmus Strecke: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

(Bedingung ...) // wie gehabt

var $z : \mathbb{R}$;

$z = 2 * m$;

return $(k * t * t) / (z)$;

dabei ist informell

var $z : \mathbb{R}$

eine *Variablenvereinbarung* (Schlüsselwort „**var**“):
die Variable z ist ab jetzt bekannt und es können
Ausdrücke mit z gebildet werden

$z = 2 * m$;

eine (*Wert-*) *Zuweisung*: der Variablen z wird durch
eine Substitution (hier: $[z/2 * m]$) ein Wert zugewiesen

return $(k * t * t) / (z)$

die Anweisung zur Ausgabe des Ergebnisses definiert
durch den Wert des Ausdrucks nach dem
Schlüsselwort „**return**“

4.3 Anweisungen

- In imperativen Programmen sind Anweisungen die elementaren Einheiten.
- Eine Anweisung steht für einen einzelnen Abarbeitungsschritt in einem Algorithmus.
- Anweisungen werden bei uns (und auch in Java) immer mit einem Semikolon „;“ abgeschlossen.
- Die einfachste Anweisung ist die *leere Anweisung*:
 – Die leere Anweisung hat keinen Effekt auf das laufende Programm, sie bewirkt nichts.
 – Oftmals benötigt man tatsächlich eine leere Anweisung, wenn von der Logik des Algorithmus nichts zu tun ist, die Programmsyntax aber eine Anweisung erfordert.
- Eine Variablenvereinbarung, z.B. **var** $z : \mathbb{R}$; ist eine Anweisung. Sie vereinbart, dass es diese Variable „jetzt gibt“ und diese ab jetzt in Ausdrücken vorkommen darf (deshalb steht danach auch ein „;“)
- Eine Zuweisung eines Ausdrucks zu einer (vorher vereinbarten) Variable, z.B. $z = 2 * m$; ist auch eine Anweisung

4.3 Anweisungen

- Die Rückgabe eines Ergebniswertes, z.B. **return** <Ausdruck>; ist eine Anweisung
- Wie bei Ausdrücken gibt es auch *bedingte Anweisungen*:
 if <Ausdruck> **then** <Anweisungsfolge1> **else** <Anweisungsfolge2> **endif**
 - Dabei ist <Ausdruck> vom Typ $\mathbb{B}$ und dessen Wert entscheidet, welcher der beiden Zweige ausgeführt wird
 - Achtung: <Anweisungsfolge1> bzw. <Anweisungsfolge2> können offenbar jeweils eine *Menge* von Anweisungen sein! (auch wieder bedingte Anweisungen)
- Beispiele:
 - Der Algorithmus ABS kann imperativ wie folgt formuliert werden
 Algorithmus *ABS1*: $\mathbb{Z} \rightarrow \mathbb{N}_0$
 if $x \geq 0$
 then return x ;
 else return $-x$;
 endif

4.3 Anweisungen

- Der Algorithmus *Mitte* kann imperativ wie folgt notiert werden

Algorithmus *Mitte1*: $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$

```
if  $x < y \wedge y < z$  then return  $y$ ;
else
  if  $x < z \wedge z < y$  then return  $z$ ;
  else
    if  $y < x \wedge x < z$  then return  $x$ ;
    else
      if  $y < z \wedge z < x$  then return  $z$ ;
      else
        if  $z < x \wedge x < y$  then return  $x$ ;
        else return  $y$ ;
      endif
    endif
  endif
endif
endif
endif
```

4.3 Anweisungen

- Um Konstruktionen wie
 Solange $r < 100$: Erhöhe ..
wie in einigen Wechselgeldalgorithmen zu schreiben, benötigen wir außerdem Wiederholungsanweisungen (Schleifen)
 - Gezählte Wiederholung
 for <Zaehler> **from** <Startwert> **to** <Endwert> **by** <Schrittweite> **do**
 <Anweisungsfolge> **endfor**
 - Bedingte Wiederholung
 while <Bedingung> **do** <Anweisungsfolge> **endwhile**
- Imperative Algorithmen werden typischerweise durch diese Arten von Anweisungen spezifiziert.

4.3 Anweisungen

- Im funktionalen Programmierparadigma werden Algorithmen als Funktionen dargestellt (z.B. der Algorithmus *ABS*).
- Das imperative Pendant dazu ist die *Prozedur* (z.B. der Algorithmus *ABS1*), die sogar ein etwas allgemeineres Konzept darstellt: Eine Funktion kann man als Prozedur bezeichnen, aber nicht jede Prozedur ist eine Funktion.
- Eine Funktion stellt „nur“ eine Abbildung von Elementen aus dem Definitionsbereich auf Elemente aus dem Bildbereich dar, es werden aber keine Werte verändert.
- Im imperativen Paradigma können Werte von Variablen verändert werden (durch Anweisungen). Dies kann Effekte auf andere Bereiche eines Programmes haben.
- Treten in einer Prozedur solche *Seiteneffekte* (oder *Nebeneffekte*) auf, kann man nicht mehr von einer Funktion sprechen.
- Eine Funktion ist also als eine Prozedur ohne Seiteneffekte.