

Zum Titel der Vorlesung:

Programmieren

Formulierung eines **Algorithmus** in einer **Programmiersprache**

Algorithmus

Beschreibung einer Vorgehensweise, wie man zu jedem aus einer *Klasse gleichartiger Probleme* eine Lösung findet

Beispiel: Quadratwurzel einer natürlichen Zahl

1. Summiere die ungeraden Zahlen, bis Summe = Parameter
2. Ergebnis: Anzahl der Summanden

Parameter: 16

$$\begin{array}{rcl} 1 & = & 1 \\ + 3 & = & 4 \\ + 5 & = & 9 \\ + 7 & = & 16 \end{array}$$

⇒ Ergebnis: 4

Parameter: legt konkretes Problem der Klasse fest

Korrektheit: zwei Begriffe:

partielle Korrektheit:

Falls Algorithmus anhält, dann ist Ergebnis eine Lösung
(obiger Algorithmus hält nicht für Parameter 17 an!)

totale Korrektheit:

Algorithmus hält immer an und ist partiell korrekt

1. Summiere die ungeraden Zahlen, bis $\text{Summe} \geq \text{Parameter}$
2. Falls $\text{Summe} = \text{Parameter}$
dann: Ergebnis: Anzahl der Summanden
sonst: keine Quadratzahl

(Details \rightsquigarrow Vorlesung „Semantik von Programmiersprachen“)

Eigenschaften von Algorithmen

- ▶ Endlichkeit der Beschreibung (\nrightarrow Terminierung!)
- ▶ Effektivität der Schritte
(jeder Einzelschritt ausführbar, terminiert)
- ▶ Determiniertheit (evtl. verzichtbar)
 - ▶ Wirkung jedes Schrittes eindeutig
 - ▶ nächster Schritt eindeutig

„Alltägliche Algorithmen“:

- ▶ Kochrezepte
- ▶ Bastelanleitungen
- ▶ Spielregeln

\rightsquigarrow *selten exakt formuliert*

Gibt es für alle (mathematisch formulierbare) Probleme Algorithmen?

Nein! Es gibt Funktionen, zu denen es keine Algorithmen gibt!

Definiere folgende Funktion:

$f : \text{Java-Programme} \rightarrow \{0, 1\}$

$$f(P) = \begin{cases} 1 & \text{falls } P \text{ anhält} \\ 0 & \text{sonst} \end{cases}$$

Hierfür gibt es keinen Algorithmus!

(Halteproblem, s. Vorlesung Theoretische Grundlagen der Informatik)

Zur Korrektheit

- ▶ sollte immer gegeben sein, aber:
- ▶ formal nicht nachprüfbar
 - ▶ Aufgabenstellung nicht präzise
 - ▶ kein allgemeingültiges Verfahren
- ▶ **Testen** kann nur Anwesenheit von Fehlern zeigen
- ▶ Beherrschung der Komplexität durch **Strukturierung**
 - ▶ Zerlege große Programme in kleinere Einheiten (**Module**)
 - ▶ Techniken zur **Komposition**

Programmiersprache

Programmiersprache: Hilfsmittel zur Formulierung von Algorithmen

- ▶ ausführbar auf einem Rechner
- ▶ Bearbeitung mit Werkzeugen
(Transformation, Bibliothek, Versionsverwaltung)

Aspekte:

Syntax: Was sind zulässige Zeichenfolgen \rightsquigarrow Bildungsgesetze

Semantik: Was bedeuten die Zeichenfolgen?

\rightsquigarrow Interpretation der einzelnen Komponenten

Pragmatik: Wie wendet man die Sprache an?

Kodierungshinweise, Systemumgebung, Debugger, . . .

Programmiersprache

Anforderungen an Programmiersprachen:

- ▶ Universalität
(Formulierung beliebiger Algorithmen)
- ▶ automatisch analysierbar
(Compiler, Interpreter)

Vorlesung Programmierung

Unwichtig:

- ▶ bestimmte Sprache bis zur letzten Feinheit lernen
- ▶ effiziente Programme schreiben
- ▶ trickreiche Programme schreiben

Wichtig:

- ▶ Gefühl für guten Stil entwickeln
- ▶ Techniken zur Beherrschung komplexer Zusammenhänge
- ▶ Denken in Strukturen und Algorithmen
- ▶ Systematische Entwicklung von Programmen

Vorlesung Programmierung

Methode:

- ▶ Anleitung zur systematischen Konstruktion von Programmen
- ▶ eigene Programme erstellen + kritische Analyse!
- ▶ Modifikation und Erweiterung von Programmen

Programmelemente

- ▶ elementare Ausdrücke (einfachste Einheiten)
- ▶ Mittel zur Kombination (Zusammensetzen von Ausdrücken)
- ▶ Mittel zur Abstraktion
(Benennung komplexer Ausdrücke, ignoriere Details)

Programmieren mit zwei Arten von Objekten:

- ▶ Prozeduren
- ▶ Daten

Programmiersprache muss

- ▶ **elementare Daten und Prozeduren** beschreiben können
- ▶ Daten und Prozeduren **kombinieren** können

Programmiersprache in der Vorlesung:

Scheme

- ▶ LISP (LISt Processor, McCarthy 1958, rekursive Gleichungen als Modelle für Rechenprozesse)
 - ▶ einfache Syntax
 - ▶ flexibel, erweiterbar
 - ▶ interaktiv
 - ▶ Datentypen (dynamisch geprüft)
 - ▶ automatische Speicherverwaltung
 - ▶ Programme \approx Daten
- ▶ ALGOL (ALGOrithmic Language)
 - ▶ Blockstruktur
 - ▶ lexikalische Variablenbindung

Scheme

Informationen: viele, z.B. www.schemers.org/

Aktueller Standard: R⁶RS (Revised⁶ Report, 2007)

Verschiedene Erweiterungen: MIT Scheme, stk, scm, **DrRacket**

Arbeitsweise mit Scheme-System: (REPL: read-eval-print loop)

1. Start
2. Warte auf Bereitzeichen ">"
3. Ausdruck eintippen
4. System wertet Ausdruck aus, druckt Ergebnis, meldet Fehler
5. zurück zu Schritt 2

Interaktive Umgebung: **DrRacket** (<http://racket-lang.org/>)

www.informatik.uni-kiel.de/~mh/lehre/Inf-Prog-WS13/ ↪

Übungen