Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Softwaretechnik

Sommersemester 2016
20.07.2017

Prof. Dr. Christoph Bockisch
Michael Körber

Klausur zur Vorlesung

# Konzepte von Programmiersprachen

---

**Wichtige Hinweise:**

- Schalten Sie, soweit noch nicht geschehen, sofort Ihr Mobiltelefon aus!
- Schalten Sie auch alle anderen medizinisch nicht notwendigen potentiellen Lärmquellen aus.
- Entfernen Sie jetzt alle Gegenstände vom Tisch, außer:
  - einen Stift (kein Rot- oder Bleistift)
  - Ihren Studentenausweis und Ihren Personalausweis/Reisepass
  - Getränke
- Schreiben Sie jetzt auf jedes Blatt in Druckbuchstaben Ihren Namen und Ihre Matrikelnummer. Blätter ohne Namen ergeben 0 Punkte und werden nicht korrigiert! Füllen Sie insbesondere auch folgende Tabelle in Druckbuchstaben aus:

| | |
|---|---|
| Vorname | |
| Nachname | |
| Matrikelnummer | |
| Fachbereich | |
| Studienfach | |
| Angestrebter Abschluss | |
| Semester | |

- Die Bearbeitungszeit beträgt **2** Stunden.
- Verwenden Sie kein zusätzliches, eigenes Papier. Um eigene Gedanken unfertig aufzuschreiben, können Sie die Rückseiten benutzen. Sollten Sie aus Platzgründen eine Lösung auf eine Rückseite schreiben müssen, so machen Sie dies unbedingt gut kenntlich. In dringenden Fällen erhalten Sie auf Anfrage zusätzliche Blätter.
- Es sind keine Hilfsmittel erlaubt. Zuwiderhandlungen führen zum Ausschluss.
- Mehrere widersprüchliche Lösungen zu einer Aufgabe werden mit 0 Punkten bewertet.
- Die erreichten Punktzahlen werden im ILIAS-Bereich dieses Kurses unter Angabe der Matrikelnummer bereitgestellt.
- Falls Sie eine Frage haben, so wenden Sie sich bitte leise an einen der Tutoren.

---

**Wird bei der Korrektur ausgefüllt:**

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $\Sigma$ |
|---|---|---|---|---|---|---|---|---|---|
| Punkte | 12 | 13 | 7 | 8 | 20 | 14 | 14 | 12 | 100 |
| Erreicht | | | | | | | | | |

| **Note, Unterschrift:** | |
|---|---|

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 1: Wissensfragen** **(6*2=12 Punkte)**

Beantworten Sie die folgenden Fragen in 1–2 kurzen Sätzen!

**a)** Was ist eine Reduktionssemantik?

**b)** Racket verwendet lexikalisches Scoping. Beim Anwenden der LOCAL-Regel, werden die lokalen Definitionen in die globale Umgebung übertragen. Wie wird hierbei lexikalisches Scoping sichergestellt?

**c)** Was versteht man unter dem Begriff Shadowing?

**d)** Was sind Magic-Numbers und warum sollten sie vermieden werden?

**e)** Was sagt das Information-Hiding Prinzip in Bezug auf Funktionen aus?

**f)** In Prolog können logische Variablen in Fakten, Regeln und Anfragen vorkommen. Wofür stehen diese Variablen in **Fakten**?

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 2: Ausdrücke, Werte & Reduktion** **(5*1+8=13 Punkte)**

Welches Ergebnis liefern die folgenden Ausdrücke? Es genügt, wenn Sie das Ergebnis notieren, die angewendeten Regeln müssen nicht aufgeschrieben werden. Tritt beim Auswerten ein Fehler auf, geben Sie die Ursache des Fehlers an.

**a)** `(+ (- 5 3) (/ (+ 3 1) (- 4 (* 2 2))))`

**b)** `(* (- (sqrt (+ 41 (* 2 4))) 5) (* 3 7))`

**c)**
```
(define x 5) (cond [(< x 5) 1] [(> x 5) 2])
```

**d)** `(string-append (number->string (+ 4 5)) (/ 81 9))`

**e)**
```
(string-append (replicate (- 10 7) "ha")
    (replicate (/ 21 7) "hi") (replicate (- 18 (* 2 9)) "ho"))
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

Reduzieren Sie das folgende Programm schrittweise und unter Angabe der verwendeten Regel! Notieren Sie dabei in jedem Schritt ausschließlich den **relevanten** Teil des Programms. Unveränderte Teile können durch „. . . “ abgekürzt werden!:

**f)**
```
(define NUM 42)
(define (my-fun x)
  (cond [(and (> x 10) (< x 20)) (- NUM x)]
        [true (+ NUM x)]))
(my-fun 5)
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 3: Entwurfsrezept zur Funktionsdefinition** **(3+4=7 Punkte)**

Geben Sie Signatur, Aufgabenbeschreibung und mindestens einen Testfall für die folgenden Funktionen an.

**a)** `(define (f x y) (+ x (string->number y)))`

**b)** `(define (d f) (lambda (x) (f (f x))))`

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 4: Algebraische Datentypen** **(3+2+3=8 Punkte)**

„Es soll ein Programm zur Darstellung zweidimensionaler geometrischer Formen geschrieben werden. Im ersten Schritt soll das Programm mit Kreisen und Rechtecken arbeiten. Ein Kreis wird beschrieben durch seinen Mittelpunkt (X- und Y-Koordinate) und einen Radius. Rechtecke werden durch Ihren linken unteren Eckpunkt (X- und Y-Koordinate), die horizontale und die vertikale Kantenlänge modelliert.“

**a)** Definieren Sie geeignete Strukturen zur Repräsentation von *Kreisen* und *Recktecken* und geben Sie eine Datendefinition für den Summentyp *Geometrie* an.

**b)** Definieren Sie den algebraischen Datentyp *Geometrie* nochmals mithilfe der `define-type`-Anweisung.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|

**c)** Implementieren Sie eine Funktion zur Berechnung der Fläche einer Geometrie basierend auf der Definition aus Aufgabenteil b). Verwenden Sie zur Fallunterscheidung das `type-case` Konstrukt.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 5: Äquivalenz** **(20 Punkte)**

Betrachten Sie die folgende Funktion zur Berechnung der Summe einer Liste von Zahlen:

```
(define (sum l)
  (cond [(empty? l) 0]
        [else (+ (first l) (sum (rest l)))]))
```

Zeigen Sie, dass gilt: `(sum (append l1 l2))` $\equiv$ `(+ (sum l1) (sum l2))`. Folgende Aussagen über `append` dürfen Sie ohne Beweis verwenden:

i `(append empty l)` $\equiv$ `l`

ii `(append (cons x l1) l2)` $\equiv$ `(cons x (append l1 l2))`

**Hinweis: Führen Sie den Beweis mit struktureller Induktion über l1. Anwendungen der Funktion `sum` können unter Angabe der entsprechenden Regeln in einem Schritt zusammengefasst werden.**

| Vorname: | Nachname: | Matrikelnummer: |
| --- | --- | --- |

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 6: Listen und Funktionen höherer Ordnung** **(3+3+4+4=14 Punkte)**

Definieren Sie die nachfolgenden **Racket**-Funktionen jeweils mithilfe einer der folgenden Funktionen höherer Ordnung:

```
; [X] (X -> Boolean) (listof X) -> (listof X)
; Extrahiert alle Elemente aus l, die das Prädikat p erfüllen
(filter p l)

; [X Y] (X -> Y) (listof X) -> (listof Y)
; Bildet alle Elemente aus l mit f ab und liefert die Liste der Ergebnisse.
(map f l)

; [X Y] (X Y -> Y) Y (listof X) -> Y
; Kombiniert alle Elemente der Liste l durch f. Die leere Liste wird
; auf base abgebildet, die Elemente werden von rechts nach links durchlaufen.
(foldr f base l)
```

**a)**
```
; Number (listof Number) -> (listof Number)
; Liefert alle Zahlen aus alon, die restlos durch x teilbar sind.
(check-expect (filter/dividable 3 '(1 2 3 4 5 6)) '(3 6))
(define (filter/dividable x alon)
```

**b)**
```
; Number (listof String) -> (listof String)
; Vervielfältigt jeden String aus alos times mal.
(check-expect (repl/all 3 '("ha" "hi" "hu")) '("hahaha" "hihihi" "huhuhu"))
(define (filter/dividable x alos)
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**c)**
```
; (listof Number) -> Number
; Subtrahiert alle Folgeelemente vom ersten Element der Liste.
; Die leere Liste wird auf 0 abgebildet.
(check-expect (subt/all '(42 3 7 9 5)) 18)
(define (subt/all alon)
```

**d)** Geben Sie eine mögliche Implementierung der Funktion `foldr` an. Greifen Sie hierbei **nicht** auf vordefinierte Funktionen höherer Ordnung zurück.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 7: Generative Rekursion und Akkumulatoren** **(3+5+2+2+2=14 Punkte)**

Die Perrin-Folge ist wie folgt definiert:

$$per(n) = \begin{cases} 3, \text{ falls } n = 0 \\ 0, \text{ falls } n = 1 \\ 2, \text{ falls } n = 2 \\ per(n-2) + per(n-3) \text{ sonst} \end{cases}$$

**a)** Definieren Sie eine **Racket**-Funktion `(per n)`, die das n-te Glied der Perrin-Folge gemäß der Definition berechnet.

**b)** Implementieren Sie die Funktion nochmals als `(per/akk n)`, welche die Berechnung unter Einsatz von Akkumulatoren beschleunigt. Jegliche Hilfsfunktionen sollen dabei **lokal** sein.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|

**c)** Begründen Sie, warum der Einsatz eines Akkumulators hier sinnvoll ist!

Betrachten nun Sie folgende Implementierung zur Berechnung der Summe einer Liste von Zahlen:

```
(define (sum/acc alon)
  (local [(define (s l acc)
            (cond [(empty? l) acc]
                  [else (s (rest l) (+ (first l) acc))]))]
    (s alon 0)))
```

**d)** Geben Sie die Akkumulatorinvariante der lokalen Funktion `s` an.

**e)** Begründen Sie, warum die lokale Funktion `s` stets terminiert.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 8: Prolog** **(2+2+4+4=12 Punkte)**

Definieren Sie die folgenden Prädikate in Prolog mit Hilfe von Rekursion und Pattern-Matching. Die Verwendung von Bibliotheks-Prädikaten aus Prolog ist dabei **nicht** gestattet. Sollten Sie Hilfprädikate benötigen, definieren Sie diese selbst.

**Hinweis: Die arithmetischen Vergleichsprädikate <, >, =<, >=, =:= (Gleichheit), =\= (Ungleichheit) werden in Prolog Infix geschrieben. Beispiel:** `?- 2 =< 3.`

**a)** `max(X,Y,M)`: M ist das Maximum der beiden Ganzzahlen X und Y.

**b)** `longer(L1,L2)`: Ist erfüllt, falls die Liste L1 länger als L2 ist.

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|

**c)** `maxl(L,M)`: M ist das Maximum der numerischen Liste L.

**d)** `zip(L1,L2,LP)`: LP ensteht durch paarweises zusammenfassen der Elemente aus L1 und L2. Die Länge von LP ist dabei das Minimum der Längen von L1 und L2.

Beispiel: `?- zip([1,3,5],[2,4,6],[[1,2],[3,4],[5,6]]) -- yes.`