J. Brauer

Nachlausur Programmierparadigmen (A107)

Quartal: (2/2018)

Name des Prüflings: **Matrikelnummer:** **Zenturie:**

Dauer: 90 Min. Seiten der *Klausur* **ohne** Deckblatt: 20 Datum: 14.5.2018

Hilfsmittel:

- Keine (auch kein Taschenrechner)

Bemerkungen:

- **Bitte prüfen Sie zunächst die Klausur (alle Teile) auf Vollständigkeit.**
- **Bitte lösen Sie nicht die Heftung.**

Es sind 90 Punkte erreichbar!

Zum Bestehen der Klausur sind 45 Punkte ausreichend!

Aufgabe:	1	2	3	4	5	6	7	8	9	Summe:
Erreichbare Punkte:	8	4	14	8	14	12	12	6	12	90
Erreichte Punkte:										

Note: _____ Prozentsatz: _____ Ergänzungsprüfung: _____

Datum: _____ Unterschrift: _____

Aufgabe 1 (8 Punkte)

	Beantworten Sie die folgenden Fragen:	stimme zu	stimme nicht zu
a)	Zuweisungen an Variablen stellen ein bedeutendes Ausdrucksmittel der funktionalen Programmierung dar.		
b)	Funktionen, die Argumente akzeptieren, bezeichnet man als Funktionen höherer Ordnung.		
c)	Rekursive Funktionen sind notwendig immer auch bedingte Funktionen.		
d)	Ein Merkmal der Entwurfsvorschriften besteht in der Ableitung der Funktionsschablone aus der Struktur der zu verarbeitenden Daten.		
e)	Die Zeit, die ein Programmierer benötigt, um eine Zeile Code zu schreiben, ist eine Konstante (unabhängig von der Programmiersprache).		
f)	In funktionalen Sprachen mit strikter Auswertungsstrategie dienen thunks dazu, verzögerte Auswertung zu erreichen		
g)	Das Ersetzungsmodell für Funktionsanwendungen ist auch auf zustandsbehaftete Prozeduren übertragbar.		
h)	Logische Programme bestehen aus Fakten, Regeln und Variablenzuweisungen.		

Jede richtige Antwort wird mit je 1 Punkt, jede falsche oder nicht gegebene Antwort mit 0 Punkten bewertet.

Aufgabe 2 (4 Punkte)

Ein fundamentales Prinzip der funktionalen Programmierung lautet:

Funktionen sind Werte erster Ordnung.

Was folgt aus diesem Prinzip für die Verwendung von Funktionen?

Aufgabe 3 (14 Punkte)

Für die folgenden Clojure-Ausdrücke bzw. Ausdruckssequenzen geben Sie jeweils Wert und Typ des Ergebnisses an, das sich aus der Auswertung des jeweils letzten Ausdrucks ergibt. Sie brauchen nur die Endergebnisse – nicht deren Ableitung – angeben. Für den **Typ** des Ergebnisses genügen Angaben wie „Zahl“, „Symbol“ oder dergleichen. Falls es sich beim Ergebnis um eine primitive (eingebaute) Funktion handelt, schreiben Sie als Wert „primitive Funktion“, falls es sich um eine benutzerdefinierte Funktion handelt, schreiben Sie einfach „Funktion“. Für den **Typ** einer Funktion geben Sie den **Vertrag** in informeller Notation (z. B. `(list-of any) -> boolean`) an.

Beispiele:

Ausdruck	**Wert**	**Typ**
`(> 4 5)`	`false`	`Boolean`
`(fn [x] (* x x)`	`Funktion`	`Zahl -> Zahl`

Hinweis: **Alle Ausdrücke lassen sich ohne Fehler auswerten.**

(3.1) (2 Punkte)

```
(let [+ -
      / *]
  (/ (+ 3 2) 1))
```

Wert:

Typ:

(3.2) (3 Punkte)

```
(def x 4)
(def y 8)
(def f
  (fn [x y]
    (let [y 6]
      (- y x))))
(f 4 5)
```

Wert:

Typ:

(3.3) (4 Punkte)

```
((fn [x / y] (/ x y)) 6 - 3)
```

Wert:

Typ:

(3.4) (5 Punkte)

```
(((fn [x]
    (fn [y]
      (cons (* x x) y)))
  2) '() )
```

Wert:

Typ:

Aufgabe 4 (8 Punkte)

Gegeben sei die folgende Funktionsdefinition:

```
(def f
  (fn [n]
    (cond
      (= n 1) 1
      :else (* n (f (- n 1))))))
```

Werten Sie den folgenden Ausdruck (durch Anwendung der formalen Auswertungsregeln) **Schritt für Schritt** aus:

```
(f 2)
```

Aufgabe 5 (14 Punkte)

Ein Produkt sei wie folgt definiert:

```
;; Ein Produkt ist ein Record
(defrecord produkt [name stueckzahl])
```

Dabei sei `name` ein Symbol und `stueckzahl` eine ganze Zahl.

(5.1) (8 Punkte) Definieren Sie eine Funktion `nachzubestellendes-produkt`, die eine Liste von Produkten `lvp` und ein Produkt `p` als Argumente erwartet. Die Stückzahlen der Produkte in `lvp` geben die Anzahl der im Lager vorhandenen Exemplare des jeweiligen Produkts, die Stückzahl in `p` gibt die Sollstückzahl für dieses Produkt an. Die Funktion `nachzubestellendes-produkt` gibt ein Produkt-Record zurück, dessen Stückzahl angibt, wieviele Exemplare des Produkts nachzubestellen sind, damit sich im Lager wieder die Sollstückzahl befindet.

Beispiele:

```
(nachzubestellendes-produkt
 (list (->produkt 'messer 120)
       (->produkt 'gabel 80)
       (->produkt 'loeffel 95)
       (->produkt 'schere 50))
 (->produkt 'gabel 100))
;; liefert: (->produkt 'gabel 20)

(nachzubestellendes-produkt
 (list (->produkt 'messer 120)
       (->produkt 'gabel 80)
       (->produkt 'loeffel 95)
       (->produkt 'schere 50))
 (->produkt 'gabel 75))
;; liefert: (->produkt 'gabel 0)

(nachzubestellendes-produkt
 (list (->produkt 'messer 120)
       (->produkt 'gabel 80)
       (->produkt 'loeffel 95)
       (->produkt 'schere 50))
 (->produkt 'teeloeffel 75))
;;liefert: (->produkt 'teeloeffel 75)
```

Hinweise:

- Sie müssen für diese Funktion keine Tests angeben.
- Wenden Sie die passenden Entwurfsvorschriften an.

(5.2) (6 Punkte) Definieren Sie eine Funktion `nachzubestellende-produkte`, die zwei Listen von Produkten (`lvp1, lvp2`) als Argumente erwartet. Die Stückzahlen der Produkte in `lvp1` geben die Anzahl der im Lager vorhandenen Exemplare des jeweiligen Produkts, die Stückzahlen in `lvp2` die jeweiligen Sollstückzahlen an. Die Funktion `nachzubestellende-produkte` gibt eine Liste mit Produkt-Records zurück, deren Stückzahlen angeben, wieviele Exemplare des Produkts nachzubestellen sind, damit sich im Lager wieder die Sollstückzahl befindet.

Beispiel:

```
(nachzubestellende-produkte
 (list (->produkt 'messer 120)
       (->produkt 'gabel 80)
       (->produkt 'loeffel 95)
       (->produkt 'schere 50))
 (list  (->produkt 'gabel 100)
        (->produkt 'loeffel 90)
        (->produkt 'teeloeffel 50)))
;; liefert:
(list (->produkt 'gabel 20)
      (->produkt 'loeffel 0)
      (->produkt 'teeloeffel 50))
```

Hinweise:

- Verwenden Sie die Funktion `nachzubestellendes-produkt` als Hilfsfunktion. **Sie können damit diese Teilaufgabe auch lösen, ohne die erste bearbeitet zu haben.**
- Sie müssen für diese Funktion keine Tests angeben.
- Wenden Sie die passenden Entwurfsvorschriften an.

Aufgabe 6 (12 Punkte)

Die Verträge von drei klassischen Funktionen höherer Ordnung lauten:

```
;; map: (list-of X) (X -> Y) -> (list-of Y)

;; reduce: (X Y -> Y) Y (list-of X) -> Y

;; filter: (X -> boolean) (list-of X) -> (list-of X)
```

(6.1) (9 Punkte) Erläutern Sie die drei Funktionen, indem Sie für jede von ihnen

1. die Wirkungsweise durch einen deutschen Satz beschreiben,
2. ein Anwendungsbeispiel aufschreiben.

(6.2) (3 Punkte) Geben Sie eine Implementierung von `map` als rekursive Funktion in Clojure an.

Aufgabe 7 (12 Punkte)

Gegeben sei folgende Funktion:

```
(def g
  (fn [n]
    (cond
      (= n 0) 0
      :else (+ (* n n n ) (g (- n 1))))))
```

Beweisen Sie mittels rekursiver Induktion, dass der Aufruf `(g n)` für jede natürliche Zahl ($n \geq 0$) den Wert $\frac{n^2 \cdot (n+1)^2}{4}$ liefert.

Aufgabe 8 (6 Punkte)

Der Zusammenhang zwischen den Temperatureinheiten *Celsius* und *Fahrenheit* kann durch die folgende Gleichung beschrieben werden:

$$9C = 5(F - 32)$$

Entwickeln Sie eine Lösung dieser Gleichung mithilfe des aus Vorlesung bekannten Constraint-propagation-Systems. Die Existenz der Basis-Bausteine für die Addition und Multiplikation darf vorausgesetzt werden.

(8.1) (3 Punkte) Implementieren Sie die Gleichung als Schaltbild aus den CPS-Grundelementen.

(8.2) (3 Punkte) Setzen Sie dieses Schaltbild in eine Clojure-Prozedur um.

Aufgabe 9 (12 Punkte)

Gegeben ist folgende Faktenbasis in Prolog:

```
maennlich(kurt). weiblich(karin).
maennlich(hans). weiblich(lisa).
maennlich(stefan). weiblich(marta).
maennlich(robert). weiblich(sina).
vater(stefan, kurt). mutter(karin, marta).
vater(hans, karin). mutter(sina, kurt).
vater(stefan, lisa).
vater(kurt, marta).
```

(9.1) Formulieren Sie Prolog-Regeln für die folgenden Verwandtschaftsbeziehungen:

i. (1 Punkt) Eine Person ist **Elternteil** einer anderen Person. Dabei sei der Ausdruck `Elternteil(E, K)` als „E ist Elternteil von K" zu interpretieren.

ii. (2 Punkte) Eine Person ist **Bruder** (auch Halbbruder) einer anderen Person.

iii. (4 Punkte) Eine Person ist **Vorfahre** einer anderen Person. Dabei sei der Ausdruck `Vorfahre(X, Y)` als „X ist Vorfahre von Y" zu interpretieren.

(9.2) Welche Antworten liefert der Prolog-Interpreter auf die folgenden Fragen:

i. (2 Punkte) `Elternteil(X, kurt)`

ii. (3 Punkte) `Vorfahre(X, marta)`