J. Brauer

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Klausur Programmierparadigmen (A107)

Quartal: (3/2017)

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

Dauer: 90 Min. Seiten der *Klausur* **ohne** Deckblatt: 19 Datum: 13.10.2017

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	12	14	12	10	10	6	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 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)

```
(def + -)
(def / *)
(+ 15 (/ 2 7))
```

Wert:

Typ:

(3.2) (3 Punkte)

```
(def a 15)
(def b 'x)
(def f
  (fn [a]
    (let [b 1]
      (- a b))))
(f 2)
```

Wert:

Typ:

(3.3) (4 Punkte)

```
((fn [x + y] (+ x y)) 5 - 4)
```

Wert:

Typ:

(3.4) (5 Punkte)

```
((fn [x]
   (fn [y]
     (+ (Math/sin x) (Math/cos y))))
 3.14159)
```

Wert:

Typ:

Aufgabe 4 (12 Punkte)

Was ist der Wert des letzten Ausdrucks in folgendem Programm? Begründen Sie Ihre Antwort z. B. durch Anwendung des Ersetzungsmodells.

```
(def schnittchen 41)
(def canapee 43)

(def haeppchen
  (fn [schnittchen canapee]
    ((fn [schnittchen]
       (- schnittchen canapee))
     schnittchen)))

(haeppchen canapee schnittchen)
```

Aufgabe 5 (14 Punkte)

(5.1) (5 Punkte) Entwickeln Sie die folgende Funktion unter Anwendung der Entwurfsvorschrift III:

Die Funktion `memrest` liefere, angewendet auf ein Symbol `sym` und eine Liste von Symbolen `lvs`, die Teilliste von `lvs`, die mit dem erstmaligen Auftreten von `sym` in `lvs` beginnt. Kommt `sym` in `lvs` nicht vor, ist das Ergebnis die leere Liste.

Beispiele:

```
(memrest 'a '(x y z)) ==> '()
(memrest 'a '(x a y a z)) ==> '(a y a z)
```

Geben Sie alle Bestandteile von Entwurfsvorschrift III an.

(5.2) (9 Punkte) Schreiben Sie eine Funktion `skelett`, die ermittelt, ob eine Liste `skl` ein Skelett einer anderen Liste `lst` ist. Eine Liste `skl` ist genau dann ein Skelett einer Liste `lst`, wenn durch Entfernung beliebiger Elemente aus `lst` die Liste `skl` entsteht. Mit anderen Worten: Die Elemente von `skl` kommen in der Reihenfolge, in der sie in `skl` auftreten, auch in `lst` vor.

Im einzelnen gelten folgende Aussagen:

1. Die leere Liste ist Skelett jeder anderen Liste (einschließlich der leeren Liste): `(skelett '() l)` liefert `true`.
2. Eine nicht-leere Liste ist niemals Skelett einer leeren Liste: `(skelett '(a b x) '())` liefert `false`.
3. Falls `(first skl)` in `lst` vorkommt, dann ist `skl` genau dann ein Skelett von `lst`, wenn `(rest skl)` ein Skelett der Teilfolge von `lst` ist, die hinter dem ersten Vorkommen von `(first skl)` in `lst` beginnt. Beispiele:
   `(skelett '(a b c ) '(x a a y b c z)) ==> true`
   `(skelett '(a b c d) '(x a a y b c z)) ==> false`

Hinweise:

1. Sie brauchen nur die reine Funktionsdefinition aufschreiben.
2. Es ist zweckmäßig, für die Funktion `skelett` die Funktion `memrest` aus Teil 1 der Aufgabe zu benutzen. Dafür dürfen sie voraussetzen, dass die Listen nur Symbole enthalten.

Aufgabe 6 (12 Punkte)

(6.1) (4 Punkte) Die Funktion

```
(def zahlenfolgeaufwaerts
  (fn [z1 z2]
    (cond
      (> z1 z2) '()
      :else (cons z1 (zahlenfolgeaufwaerts (+ z1 1) z2 )))))
```

liefert, angewendet auf zwei natürliche Zahlen $z1$ und $z2$, eine Liste mit den ganzen Zahlen aus dem Intervall $z1$ bis $z2$ (einschließlich). Falls $z1$ größer als $z2$ ist, ist die leere Liste das Resultat. Beispiele:

```
(zahlenfolgeaufwaerts 10 16) ==> (10 11 12 13 14 15 16)
(zahlenfolgeaufwaerts 3 3)   ==> (3)
(zahlenfolgeaufwaerts 4 3)   ==> ()
```

Schreiben Sie eine zweite Funktion, mit Namen `zahlenfolgeabwaerts`, die ein absteigendes Zahlenintervall erzeugt. Beispiele:

```
(zahlenfolgeabwaerts 9 7) ==> (9 8 7)
(zahlenfolgeabwaerts 3 3)   ==> (3)
(zahlenfolgeabwaerts 3 4  ==> ()
```

Hinweis: Sie brauchen nur die Funktionsdefinition aufschreiben.

(6.2) (6 Punkte) Abstrahieren Sie die beiden Funktionen `zahlenfolgeaufwaerts` und `zahlenfolgeabwaerts` zu einer Funktion (`zahlenfolge`) höherer Ordnung, mit der auf- und absteigende Zahlenfolgen erzeugt werden können.

Hinweis: Sie brauchen nur die Funktionsdefinition aufschreiben.

(6.3) (2 Punkte) Geben Sie den Vertrag von `zahlenfolge` an.

Aufgabe 7 (10 Punkte)

Gegeben sei folgende Funktion:

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

Beweisen Sie mittels rekursiver Induktion, dass der Aufruf `(f n)` für jede natürliche Zahl $n > 0$ den Wert

$$f(n) = \sum_{i=1}^{n} i^2$$

liefert.

Aufgabe 8 (10 Punkte)

Die Software-Gleichung von Larry Putnam beschreibt den phänomenologischen Zusammenhang zwischen der Produktgröße G (gemessen in *lines of code*), dem Aufwand A (gemessen in Personenstunden), einem Technologiefaktor P und der Projektdauer t:

$$G = P \cdot A^{\frac{1}{3}} \cdot t^{\frac{4}{3}}$$

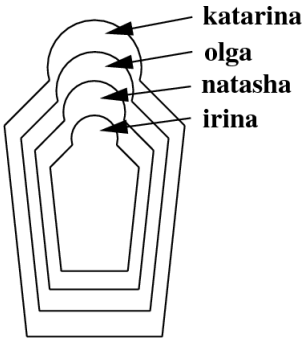
Entwickeln Sie eine Lösung dieser Gleichung mithilfe des aus Vorlesung bekannten Constraint-Programming-Systems. Die Existenz der Basis-Bausteine für die Addition, Multiplikation und Potenzbildung darf vorausgesetzt werden.

(8.1) (6 Punkte) Implementieren Sie die Gleichung als Schaltbild aus den CPS-Grundelementen.

(8.2) (4 Punkte) Setzen Sie diese Schaltbild in eine Scheme-Prozedur um.

Aufgabe 9 (6 Punkte)

Sie kennen die hölzernen, russischen Puppen – auch Babuschkas genannt –, die sich aufgrund ihrer unterschiedlichen Größe ineinander stecken lassen, wie in der folgenden schematischen Darstellung zu erkennen ist:



Schreiben Sie folgendes Prolog-Programm:

(9.1) (2 Punkte) Formulieren Sie eine Faktenbasis unter Verwendung eines Prädikats `directlyIn/2`, mit dem beschrieben wird, welche Puppe sich direkt in einer anderen Puppe befindet. Benutzen Sie die Babuschkas aus der Abbildung.

(9.2) (4 Punkte) Definieren Sie ein Prädikat `in/2`, das `true` liefert, wenn sich eine Puppe innerhalb einer anderen (direkt oder indirekt) befindet. Dieses Prädikat ist so zu schreiben, dass es auch für andere Babuschka-Konstellationen funktioniert.