2. Klausur zu 'Softwareentwicklung III:
Funktionale Programmierung'
WS 2016/2017
Leonie Dreschler-Fischer

Note:_____ und Begründung:

Hamburg,_____ Unterschrift (Prüferin)_____

Rechtsmittelbelehrung: Gegen die Bewertung dieser Prüfungsleistung kann innerhalb eines Monats nach ihrer Bekanntgabe Widerspruch erhoben werden. In diesem Zeitraum kann die Bewertung der Klausur eingesehen werden. Der Widerspruch ist schriftlich oder zur Niederschrift bei der bzw. dem Vorsitzenden des für das Hauptfach zuständigen Prüfungsausschusses einzulegen. Es wird darauf hingewiesen, dass ein erfolgloses Widerspruchsverfahren kostenpflichtig ist.

Zugelassene Hilfsmittel: Ein Wörterbuch für die deutsche Sprache, Schreibstifte.

Nicht verwendet werden dürfen: Mobiltelefone oder sonstige elektronische Kommunikationsmittel, schriftliche Aufzeichnungen zur Vorlesung, zu den Übungen und zu früheren Klausuren oder Probeklausuren.

Datum & Unterschrift Klausurteilnehmer/in: _____

Wichtige Hinweise:

- Die Klausur bitte auf jeden Fall geheftet lassen! Kein zusätzliches Papier verwenden!
- **Unterschreiben Sie die Klausur auf dieser Seite.**
- Punkte für Teilaufgaben werden durch $\{\oslash\}$ angegeben, z.B. $\{\oslash\oslash\}$ = 2 Punkte. Die Gesamtanzahl von Punkten pro Aufgabe steht jeweils im Kasten am Rand der entsprechenden Aufgabe unter "von". Beachten Sie, dass Sie für jeden zu erzielenden Punkt nur ein bis zwei Minuten Zeit zur Verfügung haben.
- Sollte der Platz für eine Aufgabe nicht ausreichen, so verwenden Sie die Leerseiten am Ende und machen Sie einen Vermerk am Rand, etwa siehe "$\Longrightarrow$ Seite 18".
- Für alle Aufgaben können Sie die Funktionen aus dem tools-module der se3-bib als vordefiniert voraussetzen.
  #lang racket
  (require se3-bib/tools-module)
- Im Anhang der Klausur, finden Sie ein kleines Handbuch zu Funktionen höherer Ordnung, CLOS und Prolog-in-Racket.

1 Ihre Klausuraufgaben

1. Zu welchen Werten evaluieren die folgenden Racket-Ausdrücke?

	von
	10

(a) `'(integer? (/ 4 2))` {⊘}

Lösung: '(integer? (/ 4 2))

(b) `(rational? (/ 6 2))` {⊘}

Lösung: #t

(c) `(car '((Hinter (eines (Baumes Rinde) wohnt die)) Made mit dem Kinde))` {⊘}

Lösung: '(Hinter (eines (Baumes Rinde) wohnt die))

(d) `(cadr '(Sie (ist Witwe denn) der Gatte den))` {⊘}

Lösung: '(ist Witwe denn)

(e) `(map car '((sie hatte) (fiel vom) (( Blatte))))` {⊘⊘}

Lösung: `'(sie fiel (Blatte))`

(f) `(foldr append '() '((diente so ) (auf) (diese Weise)))` {⊘⊘}

Lösung: `'(diente so auf diese Weise)`

(g) {⊘⊘}

```
((lambda (xs) (length xs))
    '(einer (Ameise als) Speise))
```

Lösung: 3

10 Pnkt.

von
10

2. **Reduktionsstrategien**

(a) Beschreiben Sie kurz, nach welcher Reduktionsstrategie in Racket *funktionale Ausdrücke* und „*special form expressions*“ ausgewertet werden. {⊘⊘⊘⊘}

i. Auswertung funktionaler Ausdrücke:
Lösung: Funktionale Audrücke werden strikt mit vorgezogener Auswertung reduziert.

Die Reihenfolge der Auswertung der Argumente ist undefiniert.

ii. Auswertung von „*special form expressions*“ :
Lösung: Special form expressions werden verzögert ausgewertet.
Die Reihenfolge für die Auswertung der Argumente ist spezifisch für jeden special form Operator.

(b) Geben Sie einen funktionalen Ausdruck an, für den die verzögerte Auswertung günstiger wäre als die vorgezogene Auswertung. Begründung? {⊘⊘⊘}

Lösung:
`((const 3) (sin 2))`
Das Argument (sin 2) wird nicht benötigt und würde bei verzögerter Auswertung nicht berechnet werden.

(c) Welche Rolle spielt die Bezugstransparenz bei der Auswertung funktionaler Ausdrücke und wodurch kann die Bezugstransparenz verloren gehen? Nennen Sie ein Beispiel. {⊘⊘⊘}
Lösung:
`set!, push!`

10 Pnkt.

3. **Formen der Rekursion:** Geben Sie für jede der folgenden Funktionen an, welche Formen der Rekursion vorliegen, ***und begründen Sie Ihre Antwort!***.

von
10

```
(define (diviRest a b)
  (cond ((= a b) 0)
        ((> a b) (diviRest (- a b) b))
        ((< a b) a)))

(define (teiler a b)
  ; a >= b
  (let ([ derRest (modulo a b)])
    (if (zero? derRest) b
        (teiler b (teiler b derRest)))))

(define (abbildend f xs)
  (if (null? xs)
      '()
      (append (f (car xs))
              (abbildend f (cdr xs)))))

(define (abbildListe f xss)
  (cond [(null? xss)'()]
        [(not (list? (car xss)))
         (cons (f (car xss))
               (abbildListe f (cdr xss)))]
        [else (cons (abbildListe f (car xss))
                    (abbildListe f (cdr xss)))]))

(define (dummy1? xs)
  (if (null? xs) #t (irgendwas? (cdr xs))))

(define (irgendwas? xs)
  (if (null? xs) #f (dummy1? (cdr xs))))
```

(a) `diviRest` :{⊘⊘}

Lösung: `abbildend` ist linear rekursiv und endrekursiv.

(b) `teiler` :{⊘⊘}

Lösung: `teiler` ist geschachtelt rekursiv.

(c) `abbildend` :{⊘}

Lösung: `abbildend` ist allgemein linear rekursiv.

(d) `abbildListe` :{⊘}

Lösung: `abbildListe` ist baumrekursiv.

(e) `dummy1?` :{⊘}

Lösung: `dummy1?` ist indirekt linear rekursiv.

(f) `irgendwas?` :{⊘}

Lösung: `irgendwas` ist indirekt linear rekursiv.

(g) Was ist der Vorteil der Endrekursion? {⊘⊘}

Lösung: Es wird kein Stack für Bindungsumgebungen der rekursiven Funktionsaufrufe benötigt. Der Speicherbedarf ist konstant und nicht von der Tiefe der Rekursion abhängig.

10 Pnkt.

4. **Rekursion:**

von
10

(a) Definieren Sie eine Funktion `(dritteln xs)`, die eine Liste von Zahlen als Argument erhält und auf eine neue Liste abbildet, bei der alle Zahlen durch zwei geteilt sind:
Beispiel: `(dritteln '(9 6 3))` $\longrightarrow$`'(3 2 1)`

i. als allgemein rekursive Funktion, {⊘⊘⊘}

Lösung:

```
(define (dritteln xs)
  (if (null? xs) '()
      (cons (/ (car xs) 3)
            (dritteln (cdr xs)))))
```

ii. als endrekursive Funktion, {⊘⊘⊘}

Lösung:

```
(define (dritteln-acc xs)
  (letrec ([theloop
            (lambda (xs acc)
              (if (null? xs) (reverse acc)
                  (theloop
                   (cdr xs) (cons (/ (car xs) 3) acc))))])
    (theloop xs '())))
```

iii. und mittels geeigneter Funktionen höherer Ordnung. {⊘⊘}

Lösung:

```
(define (dritteln-high-order xs)
  (map (curryr / 3) xs))
```

(b) Nennen Sie eine rekursive Datenstruktur von Racket.
Begründung! {⊘⊘}
Lösung: Eine Liste ist eine rekursive Datenstruktur, die entweder eine leere Liste ist oder ein Kopfelement, gefolgt von einer Liste.

10 Pnkt.

5. **Ein abstrakter Datentyp für Gewichtsangaben:**

Für viele Anwendungen ist es notwendig, nicht nur den numerischen Wert einer physikalischen Größe zu kennen sondern auch die Einheit, in der dieser Wert angegeben ist.

Implementieren Sie die Zugriffsfunktionen für einen Datentyp "'Gewichtsangaben"': Eine Gewichtsangabe sei repräsentiert als Liste mit zwei Elementen, nämlich dem numerischen Wert und der Einheit, beispielsweise `(0.5 kg)`, `(3 g)`, `(6.3 t)` usw.

(a) Definieren Sie eine Konstruktorfunktion `(make-weight value unit)`, die aus einem Wert und einer Gewichtseinheit ein Element vom Typ Gewichtsangabe erzeugt:
Beispiel: `(make-weight 3 'g)` ⟶ `(3 g)` {⊘}

Lösung:

```
(define (make-weight value unit)
  (list value unit))
```

(b) Definieren Sie Selektorfunktionen für den Zugriff auf den Wert einer Gewichtsangabe `(value-of-weight weight)` und die Gewichtseinheit `(unit-of-weight len)`: Beispiel: {⊘}

```
(value-of-weight (make-weight 3 'g)) ⟶ 3
(unit-of-weight (make-weight 3 'kg)) ⟶ kg
```

Lösung:

```
(define (value-of-weight w)   (car w))

(define (unit-of-weight w)   (cadr w))
```

(c) Definieren Sie eine Skalierungsfunktion `(scale-weight Gewicht fac)`, mit der Gewichtsangaben um einen bestimmtem Faktor vergrößert (verkleinert) werden können:
Beispiel: `(value-of-weight (scale-weight (make-weight 3 'g) 2)` ⟶ `6` {⊘⊘}

Lösung:

```
(define (scale-weight w fac)
  (make-weight
   (* fac (value-of-weight w))
   (unit-of-weight w)))
```

(d) Gegeben sei eine Tabelle `*conversiontable*` mit Umrechnungsfaktoren für Gewichtsangaben. Diese Tabelle sei als Assoziationsliste organisiert, die zu jeder Gewichtseinheit den Umrechnungsfaktor gegenüber der Angabe in kg enthält:

```
(define *conversiontable* ;
   '( ;  (unit . factor)
      (kg . 1)   ; Kilogramm
      (g . 0.001) ; Gramm
      (t . 1000) ; Tonne
      (ounce-US . 0.028349523) ; Amerikanische Unze
      (lb . 0.45359237) ; Pfund US
      ....
))
```

Definieren Sie die folgenden Operationen auf Gewichtsangaben unter Verwendung der Funktionen Konstruktoren und Selektoren `make-weight, unit-of-weight, value-of-weight, scale-weight`:

- Auslesen des Umrechnungsfaktors aus der Tabelle `(factor unit)`:
  Beispiel: `(factor 'g)` $\longrightarrow$ `0.001` {⊘⊘}

Lösung:

```
(define (factor unit)
  (cdr (assoc unit *conversiontable*)))
```

- Normalisierung (Wandlung in kg) `(weight->kg weight)`:
  Beispiel: `(weight->kg '(3 g))` $\longrightarrow$ `(0.003 kg)` {⊘⊘}

Lösung:

```
(define (weight->kg w)
  (make-weight
   (* (value-of-weight w) (factor (unit-of-weight w)))
   'kg))
```

- Vergleich zweier Gewichtsangaben `(weight< weight1 weight2)` und `(weight= weight1 weight2)`:
  Beispiel: {⊘⊘}
  `(weight< '(3 g) '(6 t))` ⟶ `#t`
  `(weight= '(300 lb) '(3 kg))` ⟶ `#f`

  Lösung:

```
(define (weight< w1 w2)
  (< (value-of-weight (weight->kg w1))
     (value-of-weight (weight->kg w2))))
(define (weight= w1 w2)
  (= (value-of-weight (weight->kg w1))
     (value-of-weight (weight->kg w2))))
```

- Addition und Subtraktion zweier Gewichtsangaben `(weight+ len1 len2)`.
  Beispiel:
  `(weight+ '(3 g) '(1 kg))` ⟶ `(1003.0 kg)`
  `(weight- '(1.001 t) '(1 kg))` ⟶ `(1000.0 kg)` {⊘⊘}

  Lösung:

```
(define (combine f w1 w2)
  (let ((w1kg (weight->kg w1))
        (w2kg (weight->kg w2)))
  (make-weight
   (f (value-of-weight w1kg) (value-of-weight w2kg)) 'kg)))

(define (weight+ w1 w2)
  (combine + w1 w2))

(define (weight- w1 w2)
  (combine - w1 w2))
```

(e) Gegeben sei eine Liste von Gewichtsangaben:
`( (6 t) (2 lb) (1 g) (3 ounce-US))`. Verwenden Sie Funktionen höherer Ordnung (`foldl, map, filter, iterate, curry`...), um funktionale Ausdrücke für folgende Werte zu schreiben:

- Die Abbildung der Liste auf eine Liste, die die Gewichtsangaben in Kilogramm enthält, {⊘⊘}

  Lösung:

```
(map weight->kg *example*)
```

- Eine Liste aller Gewichte, die kleiner als 1 kg sind, {⊘⊘}

  Lösung:

```
(filter (curryr weight< '(1 kg)) *example*)
```

- die Summe der Gewichtsangaben in Kilogramm, {⊘⊘}

  Lösung:

```
(apply +
       (map (compose value-of-weight  weight->kg)
            *example*))
```

- die Gewichtsangabe mit dem kleinsten Gewicht in der Liste. {⊘⊘}

  Lösung:

```
(foldl (lambda (w1 w2)
           (if (weight< w1 w2) w1 w2))
         (car *example*) (cdr *example*) )
```

20 Pnkt.

6. CLOS: Die Boutique „Kuschelwolle“ bietet Damenpullover und Jacken aus edler Wolle an, die sich frei zu Tri-Sets aus Pullover, Weste und Jacke kombinieren lassen. Modellieren Sie die Kollektion in CLOS:

von
10

(a) Definieren Sie ein hierachisches System von Klassen für die folgende Klassen: Kleidungsstück, Pullover, Jacke, Weste und Tri-Set (als Aggregat aus Pullover, Weste und Jacke). {⊘⊘⊘⊘⊘}

Die Klassen Pullover, Jacke und Weste sollen jeweils „slots“ für die Attribute Preis, Farbe und Waschtemperatur haben. Begründen Sie die Vererbungshierarchie.

Lösung:

```
(defclass* Kleidungsstück ())

(defclass* Pullover (Kleidungsstück)
         (p_preis      :accessor preis
                       :initarg :p_preis)
         (p_farbe      :accessor farbe
                       :initarg :p_farbe)
         (p_waschtemp  :accessor waschtemp
                       :initarg :p_waschtemp))

(defclass* Jacke    (Kleidungsstück)
         (j_preis      :accessor preis
                       :initarg :j_preis)
         (j_farbe      :accessor farbe
                       :initarg :j_farbe)
         (j_waschtemp  :accessor waschtemp
                       :initarg :j_waschtemp))
(defclass* Weste    (Kleidungsstück)
         (j_preis      :accessor preis
                       :initarg :w_preis)
         (j_farbe      :accessor farbe
                       :initarg :w_farbe)
         (j_waschtemp  :accessor waschtemp
                       :initarg :w_waschtemp))


(defclass* Tri-Set (Pullover Jacke Weste))
```

Begründung: Die Hierarchie wurde wie in der Aufgabe gefordert direkt modelliert. Aggregate wurden durch Mehrfachvererbung umgesetzt. Außerdem müssen die Slots eindutige Namen haben, um Uneindeutigkeiten zu vermeiden. Die Slot-Accessoren sind schon mit Sicht auf die folgende Aufgabe ausgewählt!

(b) Definieren Sie für die Klasse „Kleidungsstück“ generische Funktionen für den Zugriff auf den Preis eines Kleidungsstücks, die Farbe und die empfohlene Waschtemperatur. Verwenden Sie geeignete **Methodenkombinationen** und begründen Sie Ihren Vorschlag. {⊘⊘⊘⊘⊘}

Lösung:

```
(defgeneric* preis ((Kleidungsstück))
  :combination generic-+-combination)

(defgeneric* farbe ((Kleidungsstück))
  :combination generic-list-combination)

(defgeneric* waschtemp ((Kleidungsstück))
  :combination generic-min-combination)
```

Beim Preis sollte immer die Summe der Einzelstücke den Gesamtpreis ausdrücken, daher hier + combination. Bei der Farbe ist jede einzelne enthaltene Farbe von Bedeutung,daher sollte hier eine Liste aller Farben zurückgegeben werden. Bei der Waschtemperatur sollte immer die Temperatur zurückgegeben werden, die keins der Kleidungsstücke zerstört, also im Zweifel immer die niedrigere, auch wenn die Sachen dann evt. nicht mehr ganz sauber werden.

10 Pnkt.

von
10

7. **Prolog in Racket:** Wilma Wach beobachtet besorgt die zunehmende Verseuchung von Futtermitteln und Lebensmitteln mit Dioxin. Anhand der Hinweise von Verbraucherzentralen legt sie sich eine Datenbank mit aktuellen Meßwerten an. Sie definiert sich drei Rationen:

   Grenzwertrelation: Eine Relation „`(grenzwert lebensmittel höchstwert)`“ beschreibt die von der EU festgelegten zulässigen Grenzwerte für Dioxine (in Picogramm Dioxin pro Gramm Fettanteil) für die einzelnen Lebensmittelgruppen — für Eier ist dieser Grenzwert beispielsweise 3 pg.

   Lieferantenrelation: Eine weitere Relation beschreibt die Betriebe, die die Lebensmittel herstellen mit einem eindeutigen Schlüssel, Namen, Art des Betriebs und Ort.

   Meßwertrelation: Hier werden für Stichproben bei den einzelnen Betrieben die folgenden Angaben festgehalten: Datum der Messung, Nummer des Betriebes, das untersuchte Lebensmittel, die gemessene Belastung mit Dioxin (in Picogramm je Gramm Fett).

```
(require se3-bib/prolog/prologInRacket)
; (grenzwerte ?Lebensmittel ?grenzwert-in-pg))
(<- (grenzwerte Schweinefleisch 1))
(<- (grenzwerte Gefluegel 2))
(<- (grenzwerte Eier 3))
(<- (grenzwerte Fisch 4))
(<- (grenzwerte Leber 6))
(<- (grenzwerte Futterfett 0.75)) ; usw.

; (betrieb ?Betriebsnummer ?Name ?Art ?Ort )
(<- (betrieb 1 "Chicken␣run" Freilandeier Warendorf ))
(<- (betrieb 2 "Billiger␣Jakob" Legebatterie Minden ))
(<- (betrieb 3 "Der␣Preis␣gewinnt" Bodenhaltung Minden ))
(<- (betrieb 4 "Sauglücklich" Biohof Hamburg ))
(<- (betrieb 5 "Der␣fröhliche␣Landmann" Biohof Arcadien ))
(<- (betrieb 6 "Harlekin" Futterfette Ütersen)) ; usw.

; (messwert ?Datum ?Betriebsnummer ?Lebensmittel ?Dioxinbelastung)

(<- (messwert 11.11.2010 6 Futterfette 78))
(<- (messwert 2.1.2011 2 Eier 2))
(<- (messwert 3.12.2010 5 Eier 7))
(<- (messwert 6.12.2010 1 Eier 14))
(<- (messwert 16.12.2010 5 Schweinefleisch 4))
```

Stellen Sie die folgenden Anfragen: Verwenden Sie dabei möglichst selbsterklärende Variablennamen. Ein Minimanual zu Prolog-in-Racket finden Sie im Funktionslexikon.

(a) Wie hoch ist der zulässige Dioxingrenzwert für Eier?{⊘}

Lösung:

```
(?- (grenzwerte Eier ?gr))
```

(b) Welche Betriebe, angegeben durch den Namen und den Ort, liefern sowohl Schweinefleisch als auch Eier? {⊘⊘}

Lösung:

```
(?- (betrieb ? ?Name Freilandeier ?Ort )
    (betrieb ? ?Name Schweinefleisch ?Ort ))
```

(c) Bei welchen Betrieben, angegeben durch den Namen und den Ort, wurden Dioxinstichproben genommen? {⊘⊘}

Lösung:

```
(?- (betrieb ?Nr ?Name ? ?Ort )
    (messwert ? ?Nr ?Lebensmittel ?))
```

(d) Bei welchen Betrieben, angegeben durch den Namen und den Ort, lag die Stichprobe für den Dioxingehalt der Eier über dem zulässigen Grenzwert? {⊘⊘}

Lösung:

```
(?- (betrieb ?Nr ?Name ? ?Ort )
      (messwert ? ?Nr Eier ?Belastung)
      (grenzwerte Eier ?GW)
      (test (> ?Belastung ?GW)))
```

(e) Unifizieren Sie die folgenden Ausdrücke (falls möglich) und geben Sie für den Fall, dass die Unifikation erfolgreich war, die dabei erzeugten Variablenbindungen an. {⊘⊘⊘}

Ausdruck 1 **Ausdruck 2**	Variablenbindungen
(namen ?Name Anton) (namen Meyer ?Vorname)	**Lösung:** ?Name = Meyer ?Betrag = 5200; No.
(Haus ?C . ?REST) (?Y 2 ?C ?Y)	**Lösung:** ?REST = (2 1) ?B = 2 ?X = 1; No.

10 Pnkt.

Zusätzliche Leerseiten

Aufgabe	Punkte
1	
2	
3	
4	
5	
6	
7	
Summe:	
von	80
Bestanden?	**Ja** ◯ **Nein** ◯

Lösung:

Bestanden bei $\geq$ 37Punkten				
Punkte			Note	Bestanden?
70	–	80	1.0	Ja
66	<	70	1.3	Ja
62	<	66	1.7	Ja
59	<	62	2.0	Ja
55	<	59	2.3	Ja
51	<	55	2.7	Ja
48	<	51	3.0	Ja
44	<	48	3.3	Ja
40	<	44	3.7	Ja
37	<	40	4.0	Ja
0	<	37	5.0	Nein

Datum & Unterschrift Vorkorrektur _____

Datum & Unterschrift Prüferin _____