1. Klausur zu 'Softwareentwicklung III:
Funktionale Programmierung'
WS 2009/2010
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: Die ausgeteilten Prüfungsunterlagen, ein Handbuch zu Scheme (Revised Report on Scheme), ein Wörterbuch, Schreibstifte.

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

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 etwas weniger als 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 19".
- Für alle Aufgaben können Sie die Funktionen aus dem tools-module der se3-bib als vordefiniert voraussetzen.

```
#lang scheme
(require "se3-bib/tools-module.ss")
```

- Im Anhang der Klausur, auf Seite 17, finden Sie ein kleines Handbuch zu Funktionen höherer Ordnung, CLOS und Prolog-in-Scheme.

1 Aufgaben

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

	von
	10

(a) `(* (- 6 8) 2 0)` {⊘}

(b) `'(+ 3 1 -3)` {⊘}

(c) `(car '((Habe (nun (ach Philosophie))) Juristerei und Medizin))` {⊘}

(d) `(cadr '(und (leider auch) Theologie))` {⊘}

(e) `(map car '((durchaus studiert ) (mit heissem) (Bemuehn!)))` {⊘⊘}

(f) `(reduce-right append '((da) (steh) (ich) (nun)) '())` {⊘⊘}

(g) `((lambda (x) (< 3 x)) 4)` {⊘⊘}

von
10

2. **Reduktionsstrategien**

(a) Nach welcher Reduktionsstrategie werden in Scheme *funktionale Ausdrücke* ausgewertet?{⊘⊘}

(b) Nach welcher Reduktionsstrategie werden in Scheme *special form expressions* ausgewertet?{⊘⊘}

(c) Wann ist verzögerte Auswertung günstiger als vorgezogene Auswertung? Nennen Sie ein Beispiel.{⊘⊘}

(d) Betrachten sie diese beiden Funktionen, die Listen mit den Vielfachen der Zahl „drei“ berechnen.

Die erste Variante ist im Scheme-Dialekt „lazy“ auszuführen, der auch funktionale Ausdrücke mit verzögerter Auswertung reduziert, die zweite Variante kann mit vorgezogener Auswertung ausgeführt werden.

```
#lang lazy
(define (dreierLazy x)
  (cons x
        (dreierLazy (+ x 3))))

#lang scheme
(define (dreierEager x)
  (if (< x 3) '()
  (cons x
        (dreierEager (- x 3)))))
```

i. Was ist der Unterschied im Rekursionsschema?{⊘⊘}

ii. Warum führt der Aufruf `(dreierLazy 3)` nicht zu einer Endlosschleife?{⊘⊘}

von
10

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

```
#lang scheme
(define (mappend f xs)
  (if (null? xs)
      '()
      (append (f (car xs))
              (mappend f (cdr xs)))))

(define (maplist f xss)
  (cond ((null? xss)'())
        ((not (list? (car xss)))
         (cons (f (car xss))
               (maplist f (cdr xss))))
        (else (cons (maplist f (car xss))
                    (maplist f (cdr xss))))))

(define (ping? xs)
  (if (null? xs) #t (pong? (cdr xs))))

(define (pong? xs)
  (if (null? xs) #f (ping? (cdr xs))))

(define (ggt1 a b)
  (cond ((= a b) a)
        ((> a b) (ggt1 b (- a b)))
        ((< a b) (ggt1 a (- b a)))))

(define (ggt2 a b)
  ; a >= b
  (let (( derRest (modulo a b)))
    (if (zero? derRest) b
        (ggt2 b (ggt2 b derRest)))))
```

(a) `mappend` :{⊘}

(b) `maplist` :{⊘}

(c) `ping` :{⊘}

(d) `pong` :{⊘}

(e) `ggt1` :{⊘ ⊘}

(f) `ggt2` :{⊘ ⊘}

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

von
10

4. **Rekursion:**

(a) Kann eine Liste als eine rekursive Datenstruktur betrachtet werden? Begründung! {⊘⊘}

(b) Definieren Sie eine Funktion `(Anzahl-Von item xs)`, die eine Liste als Argument erhält und ermittelt, wie oft das Element in der Liste enthalten ist:
Beispiel: `(Anzahl-Von 1 '(1 0 1 0 1))` ⟶`3`

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

ii. als endrekursive Funktion, {⊘⊘⊘}

iii. und als Funktion höherer Ordnung. {⊘⊘}

5. **Funktionen höherer Ordnung:**

von
15

Petra Pfiffig findet beim Geländespiel einen verschlüsselten Hinweis an einen Baum geheftet. Sie vermutet, dass es sich um eine einfache Substitionschiffre nach Julius Caesar handelt, bei dem jedes Zeichen des Aphabets durch den n-ten Nachfolger im Alphabet ersetzt wird. Mit n=2 würde **A** durch **C** ersetzt, **B** durch **D** usw. Für die Zeichen am Ende des Alphabets wird modulo der Alphabetlänge gerechnet, so dass mit $n = 2$ dann **Y** auf **A** und **Z** auf **B** abgebildet wird. Um eine solche einfache Chiffre zu knacken, hilft es oft, die Buchstabenhäufigkeiten zu zählen, denn in langen deutschsprachigen Texten ist meistens **E** der häufigste Buchstabe. Dummerweise hat Petra Ihren Laptop und DrScheme nicht dabei. Helfen Sie Petra Pfiffig beim Knacken des Codes und definieren Sie geeignete Schemefunktionen zum Entschlüsseln.

Der zu entschluesselnde Text liege als String vor.

Nützliche Funktionen:

- Mit `string->list` kann ein String in eine Liste konvertiert werden.
- `char-upcase` wandelt Buchstaben in die entsprechenden Großbuchstaben, alle anderen Zeichen bleiben unverändert.
- `char=?` ist ein Vergleichsprädikat für Werte vom Typ `Char`.
- `char->integer` und `integer->char` wandeln Zeichen in deren ASCII-Code und umgekehrt.

Verwenden Sie, *wo immer möglich*, **Funktionen höherer Ordnung**.

(a) Definieren Sie eine Funktion `(buchstabenliste s)`, die den zu entschlüsselnden String in eine Liste von Elementen des Typs Char wandelt, wobei Kleinbuchstaben durch die entsprechenden Großbuchstaben ersetzt werden sollen. {⊘⊘} Beispiel:

```
(buchstabenliste "Caesar") ⟶ (#\C #\A #\E #\S #\A #\R)
```

(b) Definieren Sie eine Funktion `(haeufigkeiten chars)`, die für eine Liste von Buchstaben eine Tabelle der Buchstabenhäufigkeiten für die Buchstaben **A** bis **Z** erstellt. {⊘⊘⊘⊘}
Die Tabelle soll als Assoziationsliste repräsentiert werden. Z.B.

```
'((A . 1) (B . 3) ... (Z . 0))
```

(c) Definieren Sie eine Funktion, die für eine Liste von Buchstaben den häufigsten Buchstaben der Buchstaben **A** bis **Z** ermittelt.{⊘⊘⊘}

(d) Definieren Sie eine Funktion `knacke`, die eine Liste von Buchstaben erhält und unter der Annahme, dass der Buchstabe **E** im Klartext der Häufigste ist, anhand der Buchstabenhäufigkeiten den Schlüssel (die Verschiebung im Alphabet) errechnet.{⊘ ⊘}

(e) **Zusatzaufgabe:** Nehmen Sie eine Funktion `(rotiere ch versatz)` schon als gegeben an, die einen Buchstaben `ch` um `versatz` Positionen im Alphabet rotiert. {⊘ ⊘ ⊘ ⊘}
Definieren Sie eine Funktion `(text->klartext text)`, die einen verschlüsselten String als Eingabe erhält, mit der Funktion `knacke` den Schlüssel ermittelt und den Klartext ausgibt.

von
10

6. **Vererbung in CLOS:** Verwenden Sie für diese Aufgabe nach Möglichkeit Mehrfachvererbung und Methodenkombination.

   Eine Kurzübersicht zu den Sprachelementen von CLOS finden Sie im Anhang auf Seite 18.

   Haushaltsgeräte zum Kühlen von Lebensmitteln gibt es in vielerlei Formen: als einfache Kühlschränke ohne Gefrierfach, als Gefrierschränke oder Gefriertruhen zum Einfrieren von Speisen, als Kühlschrank mit integriertem Eisfach oder als Gefrierkombination, bestehend aus einem Kühlschrank und einem Gefrierschrank, usw. Wichtige Unterscheidungsmerkmale sind das Fassungsvermögen in Litern, der Stromverbrauch in Watt, die tiefste erreichbare Temperatur und die Abmessungen in Breite, Tiefe und Höhe.

   (a) Definieren Sie als CLOS-Klasse eine Klasse von Kühlgeräten und spezialisieren Sie diese Klasse für folgende Unterklassen: {⊘⊘⊘⊘}
   (zunächst nur die nackten Klassen ohne Attribute).

   - Kühlschrank
   - Kühlschrank mit Eisfach
   - Gefrierschrank
   - Gefrierkombination

   Begründen Sie den Entwurf.

(b) Definieren Sie für die Klasse „Kühlgeräte“ generische Funktionen, um die folgenden Werte abzufragen: {⊘⊘⊘⊘}
Fassungsvermögen, Abmessungen (Breite, Tiefe, Höhe), tiefste Temperatur und Stromverbrauch und wählen Sie die passende Methodenkombination für den Fall der Mehrfachvererbung. Nehmen Sie für die Abmessungen der Gefrierkombination an, dass der Kühlschrank über den Gefrierschrank gestapelt wird.
Begründen Sie die Auswahl der Methodenkombinationen.

(c) Erklären Sie den Begriff der Klassenpräzedenzliste und begründen Sie die Notwendigkeit einer solchen Liste anhand des obigen Beispiels. {⊘⊘}

von
10

7. **Prolog in Scheme:** Der Verein *Borussia Pecunia non olet* finanziert sich vor allem durch Spielertransfers. Die folgenden Prädikate beschreiben den Bestand an Spielern und die erfolgten Transfers.

```
(require se3-bib/prolog/prologInScheme)
; (fussballer
;   ?SpielerNummer ?Position ?Name ?Vorname ?Geburtsjahr))
(<- (fussballer 1 stuermer dribbler andi 1976))
(<- (fussballer 2 stuermer dribbler bernd 1973))
(<- (fussballer 3 stuermer turnschuh thomas 1975))
(<- (fussballer 4 verteidiger knochenbrecher hans 1973))
(<- (fussballer 5 mittelfeldspieler steinImSchuh martin 1977))
(<- (fussballer 6 torwart urwaldSchrei oliver 1985))
(<- (fussballer 7 stuermer kopfNuss zinedine 1977))


; (transfer ?Vorgangsnr ?SpielerNummer
;           ?Verkaeufer ?Kaeufer ?Preis ?Verkaufsdatum)
(<- (transfer 1 1
     svFlachhalter vfbHaeusleBauer 250000 "2004.04.01"))
(<- (transfer 2 3
     scMaurer vflLammsburg 260000 "1988.12.13"))
(<- (transfer 3 3
     vflLammsburg svFlachhalter 315000 "2001.12.01"))
(<- (transfer 4 5
     eintrachtBankenviertel MSVZebras 1500000 "1998.06.01"))
(<- (transfer 5 3
     svFlachhalter fsvMais05 400000 "2002.12.01"))
(<- (transfer 6 2
    svFlachhalterDieZweite vfbHaeusleBauer 250000 "2004.04.01"))
(<- (transfer 7 6
    vflLammsburg svFlachhalterDieZweite 315000 "2001.12.01"))
```

Stellen Sie die folgenden Anfragen: Verwenden Sie dabei möglichst selbsterklärende Variablennamen. Ein Minimanual zu Prolog-in-Scheme finden Sie auf Seite 19 dieses Klausurbogens.

(a) In welchem Jahr ist Bernd Dribbler geboren? {⊘}

(b) Welche Spieler (angegeben durch Name und Vorname) sind vor 1976 geboren? {⊘ ⊘}

(c) Welche Spieler (angegeben durch Name und Vorname) wurden an welchen Verein für mehr als 1000000€ verkauft?{⊘ ⊘}

(d) Welche Spieler haben schon mehrfach den Verein gewechselt?{⊘ ⊘}

(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
(Gehalt ?Name 5200) (Gehalt Meyer ?Betrag)	
(1 ?B . ?REST) (?X 2 ?B ?X)	

2 Funktionslexikon

2.1 Funktionen höherer Ordnung

2.1.1 Verknüpfung von Funktionen

(curry f arg1 ... argn) : Partielle Anwendung von `f` auf `arg1 ... argn` von links nach rechts

(rcurry f argm ... argn) : Partielle Anwendung von `f`, Bindung der Argumente von rechts nach links

(compose f1 ... fn) : Funktionskomposition, Hintereinanderausführung `(f1 (... (fn x)))`

(conjoin p1? ... pn?) : Konjunktion von Prädikaten

(disjoin p1? ... pn?) : Disjunktion von Prädikaten

(always x) : Die konstante Funktion, deren Wert unabhängig von den Argumenten `x` ist.

2.1.2 Idiome der funktionalen Programmierung

(apply f xs) : Anwendung einer Funktion `f` auf eine Liste von Argumenten `xs`

(map f xs1 ... xsn) : Abbilden einer oder mehrerer Listen auf die Liste der Bilder der Elemente. Die Stelligkeit der Funktion f muß mit der Anzahl der angegebenen Listen xs1...xsn usw. übereinstimmen.

(filter p? xs) : Die Liste der Elemente von `xs`, die `p?` erfüllen

(reduce f xs seed) : Paarweise Verknüpfung der Elemente von `xs` mit `f`, Startwert `seed`, `(f x1 (f x2 (... (f xn seed))))`

(iterate f end? start) : Die Liste der Funktionsanwendungen, bis `end?` erfüllt ist, `(start (f start) (f (f start)) ...)`

(untilM f end? start) : Der erste Wert der Folge `start, (f start), (f (f start)), ...` der `end?` erfüllt

(some p? xs) : Finde das erste Element von `xs`, das das Prädikat `p?` erfüllt, ansonsten gebe #f zurück.

(every p? xs) : wahr, wenn alle Elemente von `xs` das Prädikat `p?` erfüllen.

(assoc key alist) : Suche in der Assoziationsliste `alist` das erste Paar, daß das als *Kopf* den Schlüssel `key` enthält.

(rassoc key alist) : Suche in der Assoziationsliste `alist` das erste Paar, daß das als *Rest* den Schlüssel `key` enthält.

2.2 CLOS

2.2.1 Klassen

```
(defclass <Name der Klasse> ({<Oberklassen>})
  {(<Slot> {<Slot-keys>})}
  {<Class-keys>}
  )
```

Optionen für die Attribute (slots):

- `:initarg keyword: <key>` *Schlüsselwort für den Konstruktor*
- `:initializer <func>` *Initialisierungsfunktion*
- `:initvalue <value>` *Defaultwert*
- `:reader <name>` *Akzessorfunktion zum Lesen des Wertes*
- `:writer <name>` *Akzessorfunktion zum Schreiben des Wertes*
- `:accessor <name>` *Akzessorfunktion zum Lesen und Schreiben des Wertes*
- `:accessor <name>` *Akzessorfunktion zum Lesen und Schreiben des Wertes*
- `:type <type>` *Typdefinition für die Klasse des slots*

```
(defgeneric <name>({(<arg> <class>)} {<arg>})
    {:combination <combination>}
    {:documentation <string>} )

(defmethod <name> [<qualifier>]({(<arg> <class>)} {<arg>})
    {:documentation <string>} )
```

2.2.2 Ergänzungsmethoden

```
<qualifier> ::= :after | :before | :around
```

2.2.3 Methodenkombinationen

```
generic-+-combination, generic-list-combination
generic-min-combination, generic-max-combination
generic-append-combination, generic-append!-combination
generic-begin-combination, generic-and-combination
generic-or-combination
```

2.3 Prolog-in-Scheme-Lexikon

Variable: ?X : Die Namen von Variablen beginnen mit einem Fragezeichen.

Anonyme Variable: ? : Ein Fragezeichen bezeichnet eine anonyme Variable, die in Ausgaben unterdrückt wird.

Regeln: Klauseln mit Prämissen (Zielen) :

```
<Klausel-Kopf> :- <Ziel 1>...<Ziel n>
```

Negation: not : Der not-Oprator negiert eine Klausel : `(not <Klausel>)`

Ungleichheit: != : Das !=-Prädikat ist wahr, wenn zwei Strukturen oder Variablen nicht unifizieren :

```
(!= <struktur1> <struktur2> )
```

Das assert-Macro: ← : Trage eine Klausel in die Datenbasis ein.

```
(<- <Klausel>)
```

Das query-Macro: ?- : Anfrage, durch welche Variablenbindungen die Konjunktion der Ziele in der Anfrage erfüllt werden kann.

```
(?- <Ziel 1> ... <Ziel n>)
```

findall: findall sammelt alle Resultate des Prädikatsaufrufs `<Ziel>` in einer Liste `<Liste>` als instanziierte Varianten des Ausdrucks `<Term>` .

```
(findall <Term> <Ziel> <Liste>)
```

count: count zählt die Resultate des Prädikatsaufrufs `<Ziel>` und bindet die Anzahl an die Variable `<Var>`.

```
(count <Var> <Ziel>)
```

Funktionale Auswertung: is: Das `is`-Prädikat bindet den Wert eines funktionalen Ausdrucks an eine Variable. Alle Variablen des Ausdrucks müssen vorher an Werte gebunden sein.

```
(is <Var> <Ausdruck> )
```

Funktionale Auswertung: test: Das `test`-Prädikat evaluiert einen funktionalen Ausdruck. Das Prädikat ist erfüllt, wenn der Ausdruck wahr ist. Alle Variablen des Ausdrucks müssen vorher an Werte gebunden sein.

```
(test  <Ausdruck> )
```

Zusätzliche Leerseiten

Aufgabe	Punkte
1	
2	
3	
4	
5	
6	
7	
Summe:	
von	75
Bestanden?	**Ja** ◯ **Nein** ◯

Datum & Unterschrift Vorkorrektur _____

Datum & Unterschrift Prüferin _____