1. Klausur zu 'Softwareentwicklung III:
Funktionale Programmierung '
WS 2008/2009
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 etwa 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 "$\Longrightarrow$ Seite 11".

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

von
10

(a) `(* (- 6 7) 2)` {⊘}

(b) `'(+ 1 2 -3)` {⊘}

(c) `(car '(((An) de Eck) steiht een Jung))` {⊘}

(d) `(cdr '(mit (nem) trudelband))` {⊘}

(e) `(map car '((1 2) (1 4) (1 3)))` {⊘⊘}

(f) `(filter number? '( #f 42 (drei) (1 2)))` {⊘⊘}

(g) `((lambda (x) (* x x)) 2)` {⊘⊘}

von
10

2. **Notation arithmetischer Ausdrücke:**

   (a) Beschreiben sie den Unterschied zwischen Präfix- und Infixnotation für arithmetische Ausdrücke. Erklären Sie die jeweiligen **Vor**-und **Nachteile** und geben Sie für jede Notationsform ein **Beispiel** an. {⊘⊘⊘⊘⊘⊘}

   (b) Welche Notationsform wird in Scheme verwendet? {⊘}

(c) Gegen Sie geeignete Scheme-Ausdrücke an, um im Scheme-Interpreter die folgenden Werte zu berechnen:{⊘ ⊘ ⊘}

i. $2 - 3 * 4 + 3$

ii. $\sqrt{1 - \cos^2(3.1414)}$

von
10

3. **Rekursion:**

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

   (b) Definieren Sie eine Funktion zur Berechnung der Länge einer Liste:

      i. als allgemein rekursive Funktion {⊘⊘⊘}

ii. und als endrekursive Funktion. {⊘⊘⊘}

iii. Was ist der Vorteil der Endrekursion? {⊘⊘}

x

von
10

4. **Einfache Funktionen: Ein abstrakter Datentyp für Währungen**

Um auf einem Konto eingezahlte Geldbeträge verrechnen zu können, muß die Währung bekannt sein, auf die sich der Betrag bezieht. Definieren Sie einen abstrakten Datentyp, mit dem Sie Geldbeträge als Betrag mit Währungsangabe repräsentieren, beispielsweise als Liste mit zwei Elementen: `(<Betrag> <Währung>)`

(a) Definieren Sie eine Konstruktorfunktion `(make-deposit amount currency)`, die die Repräsentation `(deposit)` für einen eingezahlten Geldbetrag (amount) der Währung `(currency)` konstruiert. {⊘}
Beispiel:

```
(make-deposit 3 'EURO) ⟶ (3 EURO)
```

(b) Definieren Sie Selektorfunktionen für den Zugriff auf den Betrag einer Einzahlung `(amount-of deposit)` und die Währungsangabe `(currency-of deposit)`:
Beispiel: {⊘⊘}

```
(amount-of   (make-deposit 3 'USD)) ⟶ 3
(currency-of (make-deposit 3 'USD)) ⟶ USD; US Dollar
```

(c) Definieren Sie eine Multiplikatorfunktion `(multiply-deposit deposit fac)`, mit der Geldbeträge um einen bestimmtem Faktor vergrößert (verkleinert) werden können: Beispiel: {⊘⊘}

```
(amount-of (multiply-deposit (make-deposit 3 'USD) 2)) ⟶6
```

(d) Gegeben sei eine Tabelle der aktuellen Umrechnungskurse, implementiert als Assoziationsliste:

```
(define *Kurse*
  (; Währung .  Umrechnungsfaktor: Wieviel ist
   ;            eine Einheit der Währung in Euro Wert?
   (NOK . 0.1106265) ; Norwegische Kronen
   (USD . 0.7807)    ; US Dollar
   (SEK . 0.0932937) ; Schwedische Kronen
   ...
   (EURO . 1.0))
```

Definieren Sie eine Konversionsfunktion `deposit->Euro`, mit der Sie einen Geldbetrag in Euro umrechnen. Beispiel: {⊘⊘}

```
(deposit->Euro (make-deposit 1 'USD)) ⟶ (0.7807 EURO)
```

(e) Definieren Sie eine Funktion `(deposit+ deposit1 deposit2)` zur Addition zweier Geldbeträge. Beispiel: {⊘⊘⊘}

```
(deposit+ (make-deposit 1 'USD)
          (make-deposit 2 'EURO))          ⟶ (2.7807 EURO)
```

5. **Funktionen höherer Ordnung:**

von
10

Peter Pechvogel macht mit Freunden auf der Yacht „Andrea Doria“ einen Segeltörn von Kopenhagen nach Dublin. Als die Aufgaben an Bord verlost wurden, hat er leider den Kürzeren gezogen, und er muß die Bordkasse für die Einkäufe führen. In den angelaufenen Häfen wird in unterschiedlichen Währungen bezahlt, und so findet sich in seiner Kasse bald ein buntes Gemisch von Devisen — Euro, Dänische Kronen (DKK), Schwedische Kronen (SEK), Norwegische Kronen (NOK), Britische Pfund (GBP) und sogar US Dollar (USD). Helfen Sie ihm bei der Buchführung und schreiben Sie ihm geeignete Scheme-Funktionen zur Verrechnung der Ein- und Auszahlungen.

Der Geldbestand in der Bordkasse sei repräsentiert als Liste von Einlagen `(deposit)`, wie in Aufgabe 4 beschrieben, beispielsweise

```
(define *bordkasse*
  (list (make-deposit 10 'Euro) (make-deposit 5 'Euro)
        (make-deposit 500 'DKK) (make-deposit 200 'DKK)
        (make-deposit 20 'NOK) (make-deposit 100 'NOK)
        (make-deposit 200 'USD) (make-deposit 100 'GBP) ...
        (make-deposit 300 'USD)))
```

Definieren Sie geeignete Funktionen für die folgenden Abfragen und verwenden sie dazu *wann immer möglich* ***Funktionen höherer Ordnung***, wie `map, reduce` usw.:

(a) Was sind die Einlagen in der Bordkasse jeweils in Euro wert? Definieren Sie eine Funktion `(kasse->Euro bordkasse)`, die eine Liste der Einlagen auf eine Liste der Einlagen in Euro umrechnet. {⊘⊘}

(b) Was ist der Wert der Bordkasse insgesamt in Euro? Definieren Sie eine Funktion `(summeEuro bordkasse)`, die die Summe der Einlagen in Euro errechnet.{⊘⊘}

(c) Die Yacht ist gerade im Hafen von Bergen in Norwegen eingelaufen, und Peter Pechvogel fragt sich, wie viel Norwegische Kronen wohl noch in der Kasse sind. Definieren Sie eine Funktion `(bestand bordkasse waehrung)`, die für eine Bordkasse eine Teilliste der Einlagen erstellt, die in der betreffenden Währung eingezahlt wurden. {⊘⊘}

x x

(d) Definieren Sie eine Funktion `(summeWaehrung bordkasse waehrung)`, die die Summe des Teils der Einlagen berechnet, der in einer bestimmten Währung vorliegt. {⊘⊘}

(e) Wann ist eine Funktion eine Funktion höherer Ordnung? {⊘⊘}

6. **CLOS und generische Funktionen:**

von
10

(a) **Vererbung in CLOS:** {⊘⊘⊘⊘}
Definieren Sie als CLOS-Klasse eine Klasse von Meßgeräten (ohne Attribute, aber mit der generischen Funktion „`messwert-ablesen`“) und spezialisieren Sie diese Klasse für folgende Unterklassen:

i. Thermometer (Temperaturmesser)
ii. Barometer (Luftdruckmesser)
iii. Regenmesser
iv. Wetterstation (die sowohl Temperatur als auch Luftdruck und Regen messen kann)

Begründen Sie Ihre Klassenhierarchie!

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

(c) Nehmen Sie an, dass es noch zwei weitere generische Funktionen für Messgeräte gibt:

   i. `gib-Einheit`, welche die Einheit des gemessenen Wertes zurückgibt und
   ii. `gib-Genauigkeit`, wodurch die Genauigkeit des gemessenen Wertes im Intervall [0..1] angegeben wird.

   Erklären Sie anhand aller bisher definierten generischen Funktionen den Begriff und die verschiedenen Möglichkeiten der Methodenkombination in CLOS. {⊘⊘⊘}

von
10

7. Gustav Geizig möchte sich eine neue Kamera anschaffen und hat im Internet viele günstige Angebote gefunden. Als Entscheidungshilfe, welches Angebot für ihn das günstigste ist, hat er die Angebote relational als Prolog-in-Scheme-Datenbasis modelliert.

   - Eine Relation beschreibt die Anbieter: Anbieterkennung, Anbieternamen, die Lieferzeit, die Versandkosten und die Gewährleistungsfrist in Monaten.
   - Eine Relation beschreibt die Angebote: Was kostet ein bestimmtes Produkt bei einem bestimmten Anbieter?
   - Eine Relation beschreibt die Kameras: Kamerakennung, Kameraname, Kamerahersteller, Zoomfaktor, Auflösung in Megapixeln.

   Ein Auszug aus der Datenbasis:

```
;(anbieter kennung name lieferzeit-tage
;            versandkostenEuro garantieMonate )
(<- (anbieter anb1 "Billiger␣Jakob" 3 10 6))
(<- (anbieter anb2 "Ich␣bin␣doch␣nicht␣bloed" 2 20 12))
(<- (anbieter anb3 "Exklusiv␣Versand" 1 20 36))
(<- (anbieter anb4 "Bilbo␣Baggins" 10 20 36))

;(angebot kamerakennung anbieterkennung preis)
(<-(angebot kam1 anb1 499))
(<-(angebot kam1 anb2 512))
(<-(angebot kam1 anb3 998))
(<-(angebot kam2 anb1 698))
(<-(angebot kam2 anb2 671))
(<-(angebot kam2 anb3 699))
(<-(angebot kam3 anb4 2500))

;(kamera kamerakennung Kameraname Hersteller Zoomfaktor Auflösung)
(<-(kamera kam1 "Cybershot" "Photoprofi" 18 12))
(<-(kamera kam2 "Click-and-go" "Kein␣Handbuch␣nötig" 12 5))
(<-(kamera kam3 "Miniknips" "Hobbitwerke" 24 20))
```

   Formulieren Sie die folgenden Fragen in Prolog-in-Scheme, und stellen Sie entsprechende Anfragen an die Datenbasis:

   (a) Welche Kamera hat einen Zoomfaktor von 24? {⊘}

(b) Was kostet eine Kamera mit 18-fach Zoom? {⊘⊘}

(c) Welche Kameras bei welchen Anbietern haben höchstens eine Lieferzeit von einem Tag? {⊘⊘⊘}

(d) In den Sprachen Prolog und PrologInScheme sind funktionale Sprachelemente eingebettet. Nennen Sie ein Beispiel: {⊘⊘}

(e) Was ist zu beachten, wenn in relationalen Programmiersprachen funktionale Sprachelemente verwendet werden? Welche wichtige Eigenschaft geht verloren? {⊘⊘}

Zusätzliche Leerseiten

Funktionslexikon

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

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.

CLOS 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>} )
```

Ergänzungsmethoden

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

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
```

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> )
```

Aufgabe	Punkte
1	
2	
3	
4	
5	
6	
7	
Summe:	
von	70
Bestanden?	**Ja** ◯ **Nein** ◯

Datum & Unterschrift Vorkorrektur _____

Datum & Unterschrift Prüferin _____