Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Programming Concepts and Paradigms (PCP)

Programmierübung zu Scheme 3+4

Hauptthemen:	Funktionale Programmierung: Rekursive Datentypen, Rekursionen, Lokale Definitionen und Lexikalisches Scoping
Zeitfenster:	ca. 4-8 Lektionen

R. Diehl, HS2018

Diese Übung repetiert und vertieft den in Scheme 3 und 4 vermittelten Stoff. Gehen Sie zur Vorbereitung nochmals durch die Unterrichtsfolien durch und schauen Sie insbesondere, dass Sie alle auf den Folien angegebenen Code-Beispiele verstanden haben.

Die mit * gekennzeichneten Aufgaben müssen dem Dozenten oder Assistenten als Teil des Testats gezeigt werden.

Rekursive Datentypen

(Einstellung in DrRacket: "Intermediate Student")

1. Aufgabe

Folgende Definitionen liegen vor:

```
(define couple (list
    (cons "Adam" (cons "Eva" empty))
    (cons "Paul" (cons "Paula" empty))))

(define spec-list (list 1 (list 2 3 (list 5 7) 9)))
```

a) Welche Ergebnisse bringen folgende Funktionsaufrufe?

```
(rest (first couple))
(first (rest couple))
(rest (rest couple))
(first (first (rest couple)))
(rest (first (rest couple)))
(cons? (rest (rest couple)))
```

b) Gibt es eine Kombination von `first` und `rest` an, mit der man den Wert 7 aus der `spec-list` herausfiltern kann?

2. Aufgabe

Verallgemeinern Sie die Funktion `redoubler` (Folie Beispiel: Listen-Elemente verdoppeln) so, dass jedes Element einer Zahlen-Liste mit einem beliebigen Faktor multipliziert wird.

3. Aufgabe *

Untersuchen Sie die folgenden Funktionen. Was bewirken diese Funktionen?

a)

```
(define (f liste)
   (if (empty? liste)
      empty
      (if (empty? (rest liste))
         (first liste)
         (f (rest liste)))))
```

b)

```
(define (g liste)
   (cond
      ((empty? liste) empty)
      ((empty? (rest liste)) (first liste))
      (else
         (if (> (first liste) (g (rest liste)))
            (first liste)
            (g (rest liste))))))
```

4. Aufgabe *

Entwickeln Sie eine Funktion, die ein bestimmtes Element aus einer Liste entfernt. Die Definition der Funktion lautet:

```
(define (delete item a-list)
```

Beispiel zur Anwendung der Funktion:

```
> (delete 3 (list 1 2 3 4))
(list 1 2 4)
> (delete 'c '(a b c d))
(list 'a 'b 'd)
> (delete 'f '(a b c d))
(list 'a 'b 'c 'd)
> (delete 'f empty)
'()
> (delete 'f (list 1 2 3 4))
(list 1 2 3 4)
```

5. Aufgabe

Entwickeln Sie eine Prädikatfunktion, die feststellt, ob ein bestimmtes Element in einer Liste vorhanden ist. Die Definition der Funktion lautet:

```
(define (contains? item a-list)
```

Beispiel zur Anwendung der Prädikatfunktion:

```
> (contains? 'c '(a b c d))
true
> (contains? 'f '(a b c d))
false
```

Funktionen höherer Ordnung

(Einstellung in DrRacket: "Advanced Student")

6. Aufgabe

Die Folie 23 aus PCP-Scheme-4 zeigt die Funktion `list-filter`, um eine Liste zu untersuchen und ggf. eine Resultatliste zu erstellen.

```
(define (list-filter rel-op a-list value)
  (cond
    [(empty? a-list) empty]
    [else
      (cond
        [(rel-op (first a-list) value)
         (cons (first a-list)
         (list-filter rel-op (rest a-list) value))]
        [else (list-filter rel-op (rest a-list) value)])
    ]))
```

Die Tests wurden nur mit Listen von Zahlen durchgeführt. Testen Sie die Funktion auch für folgende Situationen

a) Suche ein bestimmtes Symbol in einer Liste von Symbolen
b) Suche ein bestimmtes Zeichen in einer Liste von Zeichen
c) Suche einen bestimmten String in einer Liste von Strings

7. Aufgabe *

Schreiben Sie eine eigene Prädikatfunktion für die Funktion `list-filter`, welche untersucht, ob es eine oder mehrere Zahlen in einer Liste von Zahlen gibt, die teilbar durch eine bestimmte (als Parameter gegebene) Zahl ist.

8. Aufgabe *

Sie kennen die Implementation des Sortierens durch Einfügen:

```
; Sortieren durch Einfügen
(define (sort-a-list num-list)
  (cond
    ((empty? num-list) empty)
    (else (insert (first num-list)
                  (sort-a-list (rest num-list))))
    ))

; Einfügen in sortierter Liste
(define (insert item a-list)
  (cond
    ((empty? a-list) (list item))
    ((<= item (first a-list)) (cons item a-list))
    (else (cons (first a-list) (insert item (rest a-list))))
    ))
```

Die `sort-a-list` Funktion sortiert nur Zahlen von klein nach gross. Ändern Sie die `sort-a-list` Funktion so, dass die Sortierrichtung bestimmt und nicht nur Zahlen sondern auch Strings sortiert werden können.

9. Aufgabe*

a) Untersuchen Sie die folgende Funktion. Was macht die Funktionen genau?

```
(define (a-op a)
  (cond
    [(>= a 0) +]
    [else -]))
```

b) Schreiben Sie eine Funktion `abs-a-plus-b`, welche die absoluten Werte von a und b zusammen zählt.

```
(define (abs-a-plus-b a b)
```

c) Erweitern Sie die Funktion `abs-a-plus-b` zu `abs-a-op-b` damit man eine beliebige Operation mit den absoluten Werten von a und b durchführen kann.