

Übungsblatt 1 - PR3 (Prof. Schramm)

Aufgabe 1

(a) Elemente tauschen

```
(defun rotiere (liste)
  (if (or (null liste) (null (cdr liste)))
      (liste)
      (append (cdr liste) (list (car liste)))
  )

;;; Alternativ: first/rest statt car/cdr
(defun rotiere_alternative (liste)
  (append (rest liste) (list (first liste)))
)
```

(b) Element einfügen

```
(defun neues-vorletztes (ele liste)
  (if (or (null liste) (null (cdr liste)))
      (liste)
      (reverse (cons (car (reverse liste)) (cons ele (cdr (reverse liste))))))
)
```

(c) Länge einer Liste berechnen

```
(defun my-length(liste)
  (if (null liste)
      0
      (+ 1 (my-length (cdr liste)))
  )
)
```

(d) Länge einer geschachtelten Liste berechnen

```
(defun my-lengthr(liste)
  (cond ((null liste) 0)
        ((atom (car liste)) (+ 1 (my-lengthr (cdr liste))))
        ((not (atom (car liste))) (+ (my-lengthr (car liste)) (my-lengthr (cdr liste))))
  )
)
```

(e) Listen umkehren

```
(defun my-reverse(liste)
  (if (null liste) liste
      (append (my-reverse(cdr liste)) (list (car liste)))
  )
)
```

(f) Geschachtelte Listen umkehren

```
(defun my-reverser (liste)
  (cond ((null liste) liste)
        ((atom (car liste)) (append (my-reverser(cdr liste)) (list (car liste))))
        ((not (atom (car liste))) (append (my-reverser(cdr liste)) (list (my-reverser (car liste))))))
  )
)
```

Aufgabe 2

(a) Darstellung eines Binärbaums CAR einer Liste repräsentiert den Elternknoten, während CDR die Kinder repräsentiert. Jeder untergeordnete Teilbaum ist eine Liste.

Beispiel:

Folgender Baum:



Wird als Liste so aussehen:

(a (b((d) (e))) (c))

(b) Baumtraversierung

```
(defun is-tree(tree)
  (cond
    ((null tree) NIL)
    (t)
  )
)

(defun left-subtree(tree)
  (cond
    ((null tree) NIL)
    ((not (listp tree)) NIL)
    (t (cadr tree))
  )
)

(defun right-subtree(tree)
  (cond
    ((null tree) NIL)
    ((not (listp tree)) NIL)
    (t (caddr tree))
  )
)

(defun pre-order(tree)
  (if
    (not (is-tree tree)) NIL
    (cons (if (not (listp tree))
              tree
              (car tree))
          (append (pre-order (left-subtree tree))
                  (pre-order (right-subtree tree))
                )
        )
  )
)

(defun in-order(tree)
  (if
    (not (is-tree tree)) NIL
    (append
      (in-order (left-subtree tree))
      (if (not (listp tree))
          (list tree)
          (list (car tree))
        )
      (in-order (right-subtree tree))
    )
  )
)

(defun post-order(tree)
  (if
    (not (is-tree tree)) NIL
    (append
      (post-order (left-subtree tree))
      (post-order (right-subtree tree))
      (if (not (listp tree))
          (list tree)
          (list (car tree))
        )
    )
  )
)
```

Tests zu Aufgabe 1

```
(defvar list1 '(eins zwei drei vier))
(defvar list2 '(eins zwei (zwei (zwei drei) eins) drei vier))
(defvar item1 'dreieinhalb)

(print "Aufgabe 1a")
(defvar ret1a1 (rotiere list1))
(print ret1a1)
(defvar ret1a2 (rotiere_alternative list1))
(print ret1a2)

(print "Aufgabe 1b")
```

```
(defvar ret1b (neues-vorletztes item1 list1))  
(print ret1b)
```

```
(print "Aufgabe 1c")  
(defvar ret1c (my-length list1))  
(print ret1c)
```

```
(print "Aufgabe 1d")  
(defvar ret1d (my-lengthr list2))  
(print ret1d)
```

```
(print "Aufgabe 1e")  
(defvar ret1e (my-reverse list2))  
(print ret1e)
```

```
(print "Aufgabe 1f")  
(defvar ret1f (my-reverser list2))  
(print ret1f)
```