Übung 1 - PR3

Lotz, Johannes
Krizki, Eugen
Ranjan, Anusan
Vladimirskij, Richard
Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Str. 10, 68163 Mannheim

Inhaltsverzeichnis

1 Aufgabe 1 1
1.1 Elemente tauschen 1
1.2 Elemente einfügen 1
1.3 Länge einer Liste berechnen 1
1.4 Länge einer geschachtelten Liste berechnen 1
1.5 Liste umkehren 2
1.6 Geschachtelte Liste umkehren . . . 2

2 Aufgabe 2 2
2.1 Darstellung eines Binärbaums . . . 2
2.2 Baumtraversierung 2

Abkürzungen 3

1. Aufgabe 1

1.1. Elemente tauschen

```
;rotiere -> accepts a list and returns a new list with
    the first element in the last position
;@lst a non-nested list
(defun rotiere (lst)
  (cond
    ((null lst) nil)
    ;append the first element of the list to the back of
        the rest of the list
    (t(append (cdr lst) (list (car lst))))
    )
  )
```

Listing 1. Methode rotiere(lst)

1.2. Elemente einfügen

```
;Assisting method -> rvrs
;rvrs -> accepts a list and returns a new list where
    the order of the elements within is reversed
;@lst a non-nested list
(defun rvrs(x)
  (cond
    ((null x) x)
    ;Recursively break down the list to the last element
        and append it back together
    (t(append (rvrs(cdr x)) (list(car x))))
    )
  )
```

```
;neues-vorletztes -> takes an element and a list and
    returns a new list
;whereby the element has been inserted into the
    secondLast position
;@elem any element (atom or cons-cell)
;@list any kind of list (nested or non)
(defun neues-vorletztes(elem lst)
  (setf revLst (rvrs lst)) ;Reverse the original list
  ;Insert the element into the second position of the
     reversed list and reverse it back again
  (rvrs (append (list(car revLst)) (list elem) (cdr
     revLst)))
  )
```

Listing 2. Methoden rvrs(x) und neues-vorletztes(elem list)

1.3. Länge einer Liste berechnen

```
;my-length -> takes a list and returns its length as a
    number
;@lst a non-nested list
(defun my-length (lst)
  (cond
    ((null lst)0)
    ;Recursively break down the list building the sum of
        function calls
    (t(+ 1 (my-length(cdr lst))))
    )
  )
```

Listing 3. Methode my-length(lst)

1.4. Länge einer geschachtelten Liste berechnen

```
;my-lengthR -> takes any list (nested or not) and
    returns the amount of atoms within it as a number.
;@lst any list
(defun my-lengthR (lst)
  (cond
    ((null lst) 0)
    ;If a nested list is encountered as an element in
       the current list
    ;it needs to be processed as a "new" list apart from
       the rest as we don't know
    ;what comes next. This leads to forked recursion
       paths so the results
    ;need to be summed up
    ((listp (car lst)) (+ (my-lengthR (car lst)) (
       my-lengthR(cdr lst))))
    ;If the next element is an atom, sum it up
    (t(+ 1 (my-lengthR (cdr lst))))
```

```
    )
  )
```

Listing 4. Methode my-lengthR(lst)

1.5. Liste umkehren

```
;my-reverse -> accepts a list of atoms and returns a
    new list with the order of elements reversed
;@lst a non-nested list
(defun my-reverse(lst)
  (cond
    ((null lst) lst)
    ;Recursively break down the list to the last atom
       and append it back together
    (t(append (my-reverse (cdr lst)) (list (car lst))))
    )
  )
```

Listing 5. Methode my-reverse(lst)

1.6. Geschachtelte Liste umkehren

```
;my-reverseR -> accespts any list (nested or not) and
    returns a new list with all elements reversed
;@lst any list
(defun my-reverseR(lst)
  (cond
    ((null lst) lst)
    ;When a list-element is encountered the list is
       broken down recursively and
    ;appended back together as a new list resulting in a
        reversed order
    ((listp (car lst))(append (my-reverseR(cdr lst)) (
       list(my-reverseR(car lst)))))
    ;Same as above, just dealing with atoms.
    (t(append (my-reverseR (cdr lst)) (list (car lst))))
  )
)
```

Listing 6. Methode my-reverseR(lst)

2. Aufgabe 2

2.1. Darstellung eines Binärbaums

Ein Binärbaum wird in List Processor (LISP) als eine Liste aus Listen dargestellt. Diese Liste hat folgendes Format: Das erste Element beinhaltet einen atomaren Wert des Knoten. Die folgenden zwei Elemente stellen den linken und rechten Teilbaum dar. Die Teilbäume haben wiederum das bereits beschriebene Format. In welcher Reihenfolge Element, linker Teilbaum, rechter Teilbaum dargestellt werden, ist irrelevant. Wir haben uns für das Format Element, linker Teilbaum, rechter Teilbaum entschieden. Nachfolgend ein Beispiel:

```
(setf binary_tree '(25 (15 (10 (4) (12)) (22)) (50 (35
    (31) (44)) (70 (66) (90)))))
```

Listing 7. Binärbaum als Liste

Weiterhin zeigt die Abbildung 1 die interne Struktur des Binärbaums.

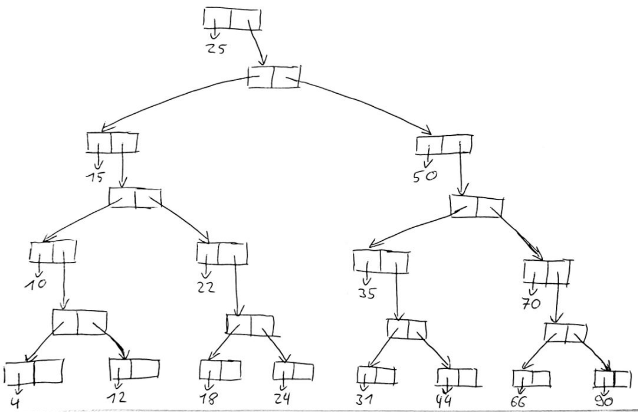

Abbildung 1. Interne Darstellung des Baums

2.2. Baumtraversierung

```
;inorder -> recursively traveses a binary tree and
    outputs the node values as a list inorder
;@tree a list that conforms to the definition of the
    binary tree (as in exercise 2A)
(defun inorder (tree)
  (cond
    ((null tree) tree);empty tree
    ((listp (car tree))(inorder(car tree)));encountering
        a branch
    ((null (cdr tree)) (append (list(car tree)))) ;
       encountering a leaf
    (t(append (inorder (cadr tree)) (list(car tree)) (
       inorder(caddr tree)))) ;encountering a node (
       Left / Node / Right)
  )
)
```

Listing 8. Methode inorder(tree)

```
;postorder -> recursively traveses a binary tree and
    outputs the node values in apostorder
;@tree -> a list that conforms to the definition of the
     binary tree (as in exercise 2A)
(defun postorder (tree)
  (cond
    ((null tree) tree)
    ((listp (car tree))(postorder(cdr tree)));
       encountering a branch
    ((null (cdr tree)) (append (list(car tree)))) ;
       encountering a leaf
    (t(append (postorder (cadr tree)) (postorder(caddr
       tree)) (list (car tree)))) ;encountering a node
       (Left / Right / Node)
  )
)
```

Listing 9. Methode postorder(tree)

```
;preorder -> recursively traveses a binary tree and
    outputs the node values in a preorder
;@tree -> a list that conforms to the definition of the
     binary tree (as in exercise 2A)
(defun preorder (tree)
  (cond
```

```
    ((null tree) tree)
    ((listp (car tree))(preorder(cdr tree)));
       encountering a branch
    ((null (cdr tree)) (append (list(car tree)))) ;
       encountering a leaf
    (t(append (list (car tree)) (preorder (cadr tree)) (
       preorder(caddr tree)) ));encountering a node (
       Node / Left / Right)
  )
)
```

Listing 10. Methode preorder(tree)

Abkürzungen

LISP List Processor