1514966 Fernando Azevedo

1330738 Florian Hrycaj

Übungsblatt 2 - PR3 (Prof. Schramm)

Aufgabe 1

insert

```
; Knoten in baum einfügen
(defun insert(tree node)
  (cond ((or (null node)(contains tree node)) nil)
        ((isEmpty tree) (list node)) ; falls baum leer
        (T (insertHelp tree node))))

; Hilfsfunktion insert
(defun insertHelp(tree node)
        ; Fall dass position zurück kommt
  (cond ((isEmpty tree) node)
        ; Fall das eltern knoten zurück kommt
        ((and (atom tree)(> tree node)) (list tree node '()))
        ((and (atom tree)(< tree node)) (list tree '() node))
        ; läuft durch den baum
        ((> (car tree) node) (list (car tree) (insertHelp (cadr tree) node)(caddr tree)))
        ((< (car tree) node) (list (car tree)(cadr tree)(insertHelp (caddr tree) node)))))
```

insertTreeFromFile

```
; Fügt Werte aus einer Datei ein
(defun insertTreeFromFile(tree filename)
    (addAll tree (getFile filename)))

; Hilfsfunktion für insertTreeFromFile
(defun getFile(filename)
  (with-open-file (stream filename
                    :direction :input
                    :if-does-not-exist nil)
    (loop for line = (read stream nil 'eof)
        until(eql line 'eof)
        collect line)))
```

contains

```
; Prüft auf Gleichheit; nil heißt nein; T heißt ja
(defun contains(tree node)
  (cond ((or (isEmpty tree)(null node)) nil)
        ((and (atom tree)(= tree node)) T)
        ((and (atom tree)(not(= tree node))) NIL)
        ( T (bTreeWalkThrough #'contains tree node))))

; Läuft durch baum
(defun bTreeWalkThrough(fn tree node)
  (cond ((= (car tree) node) (funcall fn (car tree) node))
        ((> (car tree) node) (funcall fn (cadr tree) node))
        ((< (car tree) node) (funcall fn (caddr tree) node))))
```

size

```
; Gibt Anzahl der Knoten zurück
(defun size(tree)
  (cond ((null tree) 0)
        ((atom (car tree)) (+ 1 (my-lengthr (cdr tree))))
        ((not (atom (car tree))) (+ (my-lengthr (car tree)) (my-lengthr (cdr tree))))))

; Hilfsfunktion für size
(defun my-lengthr(l)
  (cond ((null l) 0)
        ((atom (car l)) (+ 1 (my-lengthr (cdr l))))
        ((not (atom (car l))) (+ (my-lengthr (car l)) (my-lengthr (cdr l))))))
```

getMax

```
; Gibt maximalen Wert zurück
(defun getMax(tree)
  (cond ((isEmpty tree) nil)
        ((atom tree) tree)
        ((null (caddr tree)) (car tree))
        ((not(null (caddr tree)))(getMax (caddr tree)))))
```

getMin

```
; Gibt minimalen Wert zurück
(defun getMin(tree)
  (cond ((isEmpty tree) nil)
        ((atom tree) tree)
        ((null (cadr tree)) (car tree))
        ((not(null (cadr tree)))(getMin (cadr tree)))))
```

remove

```
; Löscht Wert aus Baum
(defun myremove(tree node)
  (cond ((not(contains tree node)) nil)
        ; der zustand vor dem finden
        ((and (atom tree)(= tree node)) nil)
        ; Fall das eltern knoten zurück kommt
        ; liste -> kleinstes element, ganzer linker teilbaum,
        ((= (car tree) node) (if (null (caddr tree))(cadr tree)(list (getMin (caddr tree))(cadr tree)(hilfsRemove (caddr tree) (getMin
(caddr tree))))))
        ((> (car tree) node) (list (car tree)(myremove (cadr tree) node)(caddr tree)))
        ((< (car tree) node) (list (car tree)(cadr tree)(myremove (caddr tree) node)))))

; Hilfsfunktion löscht Element eines Baums
(defun hilfsRemove(tree node)
  (cond ((atom tree) nil)
        ((= (car tree) node) (list (caddr tree)))
        ((> (car tree) node) (list (car tree)(hilfsRemove(cadr tree) node)(caddr tree)))
        ((< (car tree) node) (list (car tree)(cadr tree)(hilfsRemove(caddr tree) node)))))
```

isEmpty

```
; Schaut ob Baum leer ist
(defun isEmpty(tree) (null tree))
```

addAll

```
; Fügt Elemente des übergebenen Baums in den Vorhandenen ein
(defun addAll(tree otherTree)
  (if (car otherTree)
    (addAll (insert tree (car otherTree)) (cdr otherTree)) tree))
```

height

```
; Gibt die Höhe des Baums zurück
(defun height(tree)
  (check-type tree list)
  (if (null tree)
    0 ; empty tree
    (max (height-helper (car tree))
      (height (cdr tree)))))

; Hilfsfunktion für height
(defun height-helper(tree)
(if (atom tree)
  1 ; tree without leafs has height of 1
  (1+ (height tree))))
```

printLevelOrder

```
; Gibt den Baum in Levelorder aus
(defun printLevelOrder(tree)
    (loop while (not (null tree))
      do
        (setq node (car tree) tree (cdr tree))
        (if (not (null (car node)))
          (print (car node))
        )
        (setq tree (append tree (cdr node)))))
```