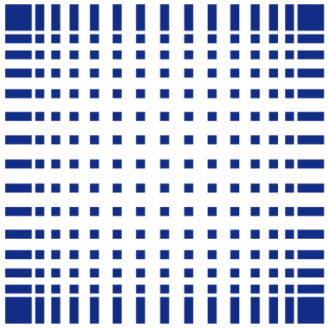

hochschule mannheim

PR3 WS 2017/18

Aufgabenblatt Nr.2

Datum: 14.12.2017
Can Arsoy (1530588)
Nico Gensheimer (1613397)
Fabian Munzinger (1529702)

Inhaltsverzeichnis

A) Insert (tree val)

```
;insert a value "val" into tree "tree"
(defun insert (tree val)
    (cond ((null tree) (list val nil nil))
          ((eql val (car tree)) (print "already exist") tree)
          ((< val (car tree)) (list (car tree) (insert (cadr tree) val) (caddr tree)))
          (t (list (car tree) (cadr tree) (insert (caddr tree) val)))
    )
)
;Example:
;existing "tree" = (5(3 (1) (4))(8 (7) (9)))
;the insert-value = 2
;Expected result: (5 (3 (1 () (2))(4))(8 (7) (9)))
;result: (5 (3(1 () (2)) (4))(8 (7)(9)))
;commands:
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(insert biba 2)
```

B) Insert (tree filename)

```
;insert all values from a file from source "filename"
(defun insert (tree filename)
  * (with-open-file (stream filename)
      (do ((char (read stream nil)
          (read stream nil)))
          ((null char))
          (append tree (list (insertNode tree char)))))
          (print tree)
)
;Example:
;existing "tree" = (5(3 (1) (4))(8 (7) (9)))
;existing "filename" = "" --> Values (8 4 12 2 6 10 14 1 3 5 7 9 11 13 15)
;Expected result: (5(3 (1 () (2)) (4))(8 (7 (6)())(9 () (12 (10 () (11))(14 (13) (15))))))
;result: (5(3 (1 () (2)) (4))(8 (7 (6)())(9 () (12 (10 () (11))(14 (13) (15))))))
```

C) Containes (tree val)

```
;containes checks, if a tree "tree" contains a value "val"
;return: T if tree contains val, 0 if tree doesn't contain val.
(defun containes (tree val)
    (cond
      ((null tree)  0)
      ((eql (car tree) val) t)
      (t
        (cond
          ((listp (car tree)) (containes (append (car tree) (cdr tree)) val))
          ((containes (cdr tree) val))
        )
      )
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;value to check: 4 and 10
;expected result from 4: T
;expected result from 10: 0
;result from 4: T
;result from 10: 0
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(containes biba 4)
;(containes biba 10)
```

D) Size (tree)

```
;size returns the amount of nodes in tree "tree"
(defun size (tree)
    (cond
      ((null tree)  0)
      (t (+ (cond
        ((listp (car tree)) (size (append (car tree) (cdr tree))))
        ((+ 1 (size (cdr tree))))
             )
          )
       )
     )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result: 7
;result: 7
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(size biba)
```

E) Height (tree)

```
;height returns the height of tree "tree"
(defun height (tree)
    (cond
      ((atom tree) 0)
      ((null tree) 0)
      (t
        (+ 1 (max (height (cadr tree)) (height (caddr tree))))
      )
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result: 3
;result: 3
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(height biba)
```

F) getMax (tree)

```
;getMax returns the biggest value of tree "tree"
(defun getMax (Tree)
    (cond
      ((equal nil (caddr Tree)) Tree)
      (t (getMax (caddr Tree)))
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result: 9
;result: 9
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(getMax biba)
```

G) getMin (tree)

```
;getMin returns the lowest value of tree "tree"
(defun getMin (Tree)
    (cond
      ((equal nil (cadr Tree)) Tree)
      (t (getMin (cadr Tree)))
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result: 1
;result: 1
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(getMin biba)
```

H) remove (tree val)

```
;remove a value "val" from tree "tree"
(defun removeN (Tree val)
  (cond
    ((equal 0 (containes Tree val)) Tree)
    (t (removeNode Tree val)
    )
  )
)
(defun removeNode (Tree val)
  (cond
    ((null Tree) nil)
    ((eql val (car Tree)) (cond
      ;
      ((and (null (cadr Tree)) (null (caddr Tree))) '())
      ((and (equal nil (cadr Tree)) (listp (caddr Tree))) (caddr Tree) (print "A"))
      ((and (listp (cadr Tree)) (equal nil (caddr Tree))) (cadr Tree) (print "B"))
      (t (append (getMin (caddr Tree)) (list (cadr Tree)) (append (list (removeNode (caddr Tree) (car (getMin (caddr Tree)))))))
                        )
    )
       ((> val (car Tree)) (append (list (car Tree)) (list (cadr Tree)) (list (removeNode (caddr Tree) val))))
       ((< val (car Tree)) (append (list (car Tree)) (list (removeNode (cadr Tree) val)) (list (caddr Tree))))
    )
)
;Example:
;existing "tree" = (5(3 (1) (4))(8 (7) (9)))
;the remove-value = 4
;Expected result: (5 (3 (1) ())(8 (7) (9)))
;result: (5 (3 (1) ())(8 (7)(9)))
;commands:
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(removeN biba 4)
```

I) isEmpty (tree)

```
;isEmpty checks tree "tree", if it is isEmpty
;return: T if null, nil if it is not null
(defun isEmpty (tree)
    (cond ((null tree) T)
      (t nil)
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result from tree: nil
;expected result from (): T
;result from tree: nil
;result from (): T
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(isEmpty biba)
```

J) addAll (tree otherTree)

```
;addAll inserts all values of "otherTree" into "tree"
(defun addAll (tree otherTree)
    (cond
      ((null otherTree) tree)
      ((null tree) otherTree)
      (t
        (setq tree (insert tree (car otherTree)))
        (setq tree (addAll tree (cadr  otherTree)))
        (setq tree (addAll tree (caddr otherTree)))
      )
    )
)
;Example
;existing tree1: (8 (2) (10))
;existing tree2: (5 (1) (11))
;expected result (tree1 tree2): (8 (2 (1) (5))(10 () (11)))
;result: (8 (2 (1)(5))(10 () (11)))
;commands
;(setq tree1 (8 () ()))
;(setq tree2 (5 (1) (10)))
;(addAll tree1 tree2)
```

K) printLevelorder (tree)

```
;printlevelorder prints the tree "tree" in traversal levelorder
(defun printlevelorder (tree)
    (levelorder (list tree))
)
(defun levelOrder (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)))
    )
)
;Example
;existing tree: (5(3 (1) (4))(8 (7) (9)))
;expected result: (5 3 8 1 4 7 9)
;result: (5 3 8 1 4 7 9)
;commands
;(setq biba '(5(3 (1) (4))(8 (7) (9))))
;(printlevelorder biba)
```