

**PR3 WS 2017/18**

Aufgabenblatt Nr.1

Datum: 27.11.2017

Can Arsoy (1530588)

Nico Gensheimer (1613397)

Fabian Munzinger (1529702)

Inhaltsverzeichnis

**Aufgabe 13**

a)3

b)3

c)..…………………………………………………………………………………………………………………………………………………..3

d).…………………………………………………………………………………………………………………………………………………..4

e).…………………………………………………………………………………………………………………………………………………..4

f)….…………………………………………………………………………………………………………………………………………………4

**Aufgabe 25**

A)5

B)6

Aufgabe 1 (Text in Grau sind Kommentare)

1. Elemente tauschen („rotiere“)

;Funktion „rotiere“ nimmt das erste Element und hängt es an das ;Ende der Liste.

(defun rotiere (listOne)

(append

(cdr listOne) (list (car listOne)))

1. Element einfügen („neues-vorletztes)

;Funktion „neues-vorletztes“ fügt das erste Element der

;übergebenen Liste „listOne“ an die vorletzte Stelle der Liste an.

;Ich nutze hier meine unten definierte Funktion „my-reverse“.

(defun neues-vorletztes (listOne)

;Mit append füge ich die neue Liste von cons ans Ende vom ;Rest der Liste. Cons erzeugt eine Liste mit dem ersten

;und letzten Element.

(append

(my-reverse (cdr (my-reverse (cdr listOne))))

(cons (car listOne) (last listOne)))

1. Länge der Liste berechnen („my-length“)

;Funktion „my-length“ gibt die Länge der übergebenen Liste ;zurück

(defun my-length (listOne)

(cond

;Wenn die übergebene Liste leer ist wird 0 ;zurückgegeben.

((null listOne)

0)

(T (+ 1 (my-length (cdr listOne))))))

1. Länge einer geschachtelten Liste berechnen („my-lengthR)

(defun my-lengthR (listOne)

(cond

((null listOne) 0)

(t (+ (cond ((listp (car listOne))

(my-lengthR (append (car listOne) (cdr listOne))))

((+ 1 (my-lengthR (cdr listOne)))))))))

E) Liste umkehren („my-reverse“)

;Funktion „my-reverse“ dreht eine Liste komplett um.

(defun my-reverse (listOne)

(cond

;Wenn „listOne“ keine Elemente hat, wird die leere ;Liste zurückgegeben

((null listOne) nil)

;Rekursiv wird immer das erste Element der ;übergebenen Liste genommen und beim ;„zurücklaufen“ wird somit zuerst das letzte ;Element genommen, dann das vorletzte usw.

(T (append

(my-reverse (cdr listOne))

(list (car listOne))))))

F) Geschachtelte Liste umkehren („my-reverseR“)

(defun my-reverseR (listOne)

(cond ((null listOne) ‘())

((listp (car listOne))

(append (my-reverseR (cdr listOne))

(list (my-reverseR (car listOne)))))

(t (append (my-reverseR (cdr listOne))

(list (car listOne))))))

Aufgabe 2

1. Darstellung eines Binärbaums

Die Darstellung eines Binärbaums wäre möglich mit geschachtelten Listen. Folgendes Beispiel von einer geschachtelten Liste ist ein Binärbaum:

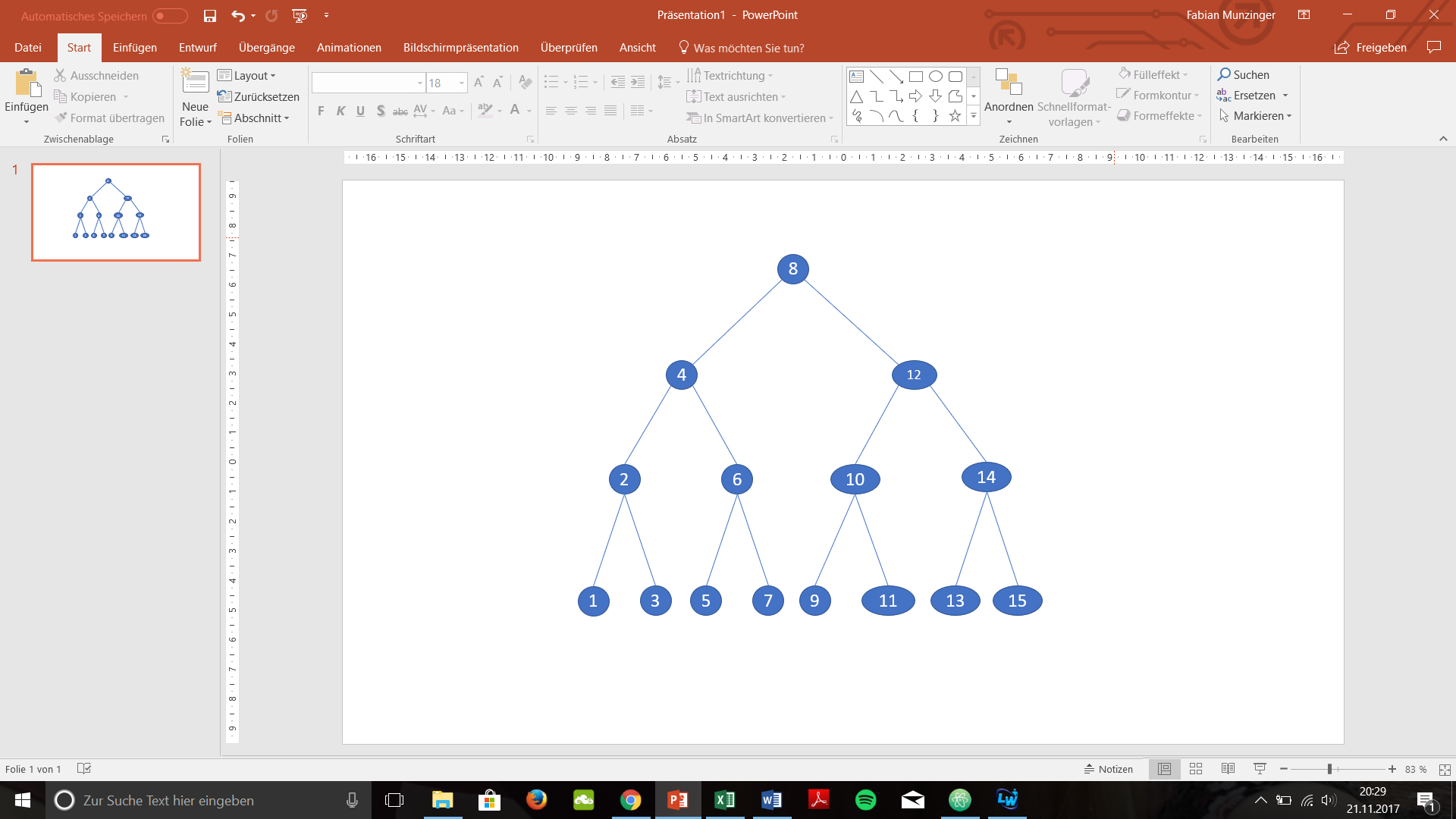
(setq biba ‘(8 (4 (2 (1) (3))

(6 (5) (7))

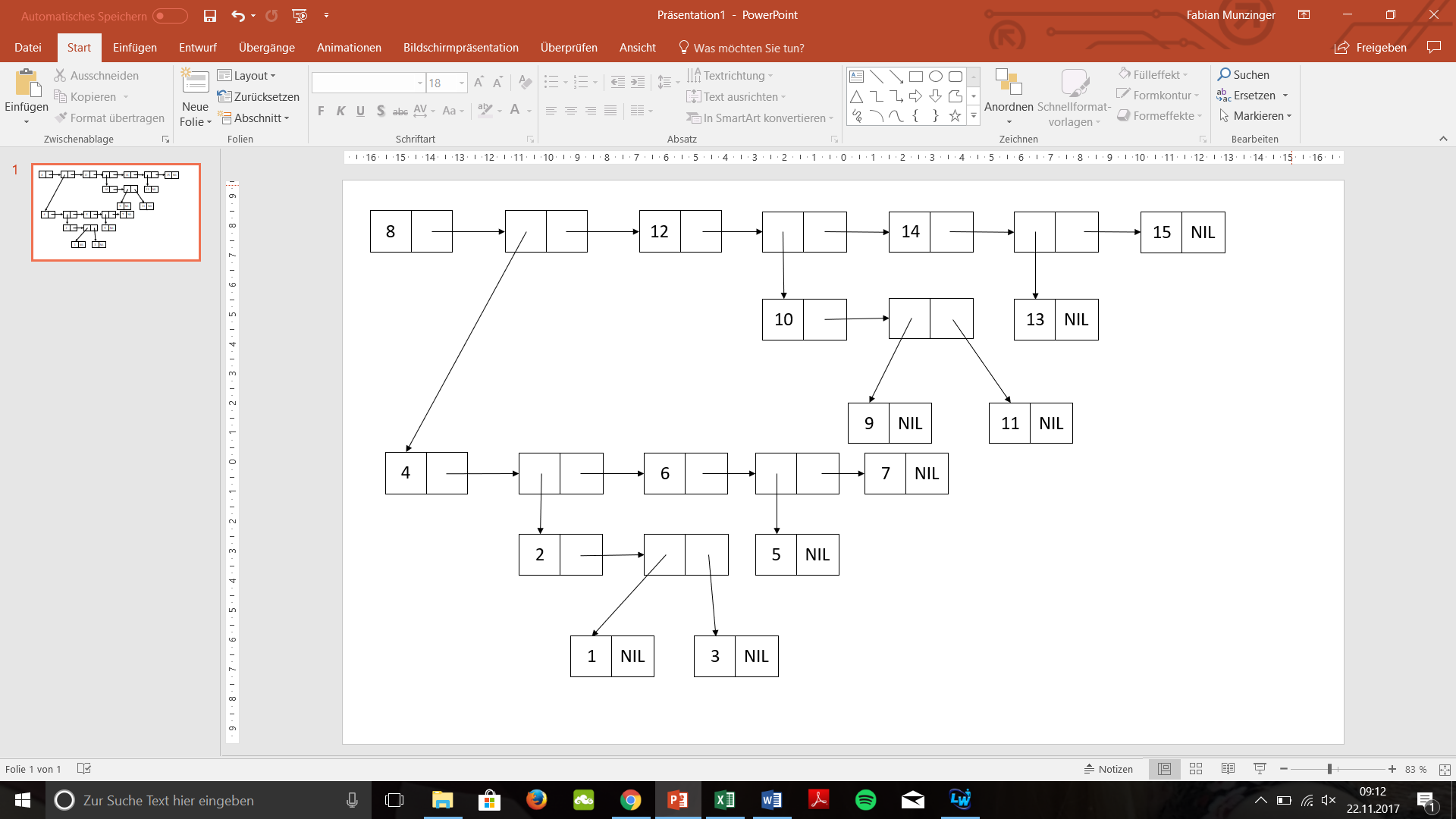
(12 (10 (9) (11))

(14 (13) (15))))))

Der Binärbaum, der hier deklariert wird, sieht wie folgt aus:



Die Listendarstellung des Baumes:



1. Baumtravesierung

Hilfsfunktionen:

;Funktion „first-node“ gibt den ersten Knoten zurück

(defun first-node (tree)

(car tree))

;Funktion „left-node“ liefert den linken Knoten zurück

(defun left-node (tree)

(cadr tree))

;Funktion „right-node“ liefert den rechten Knoten zurück

(defun right-node (tree)

(caddr tree))

* Inorder

(defun inorder (tree)

(cond ((null tree))

(t (inorder (left-node tree))

(print (first-node tree))

(inorder (right-node tree)))))

* Postorder

(defun postorder (tree)

(cond ((null tree))

(t (postorder (left-node tree))

(postorder (right-node tree))

(print (first-node tree)))))

* Preorder

(defun preorder (tree)

(cond ((null tree))

(t (print (first-node tree))

(preorder (left-node tree))

(preorder (right-node tree)))))