Maximilian Petri, 405602
Danje Petersen, 379748

Hausaufgabe 11

Aufgabe 2)

```
import Text.Show.Functions
--a)
data BinTree a b =  Blatt b | Node a (BinTree a b) (BinTree a b) deriving Show

--b)
example :: BinTree (Int -> Bool) Char
example =
    Node (\x -> x>4)
        (Node (\x -> (x*x==x))
            (Blatt 'g')
            (Node (\x -> x==0)
                (Blatt 'u')
                (Blatt 'l')
            )
        )
        (Node (\x -> x >= 7)
            (Blatt 'f')
            (Blatt 'i')
        )


--c)
countInnerNodes :: BinTree a b -> Int
countInnerNodes (Blatt b) = 0
countInnerNodes (Node _ c d) = 1 + (countInnerNodes c) + (countInnerNodes d)


--d)
decodeInt :: BinTree (Int -> Bool) b -> Int -> b
decodeInt (Blatt b) x = b
decodeInt (Node a c d) x    | a x == False = decodeInt c x
                            | a x == True = decodeInt d x


--e)
decode :: BinTree (Int -> Bool) b -> [Int] -> [b]
decode _ [] = []
decode (Node a c d) xs = map (decodeInt (Node a c d)) xs
-- ALTERNATIV:          = (decodeInt (Node a c d) x) : (decode (Node a c d) xs)


--f)
mapTree :: (b -> c) -> BinTree a b -> BinTree a c
mapTree f (Blatt v) = Blatt (f (v))
mapTree f (Node x y z) = Node x (mapTree f y) (mapTree f z)
```

Aufgabe 4)

- **i)** **f :: [Int] → Int → [Bool] → [Int]**
- **ii)** **g :: a → [a] → [a]**
- **iii)** **h :: [Int] → Int → [Bool] → [Bool] → Bool**
- **iv)** **i :: X Int Int → Int → Int → Bool**
- **v)** **j :: Bool → Bool → [Bool]**

Aufgabe 6)

```haskell
data List a = Nil | Cons a ( List a ) deriving Show


list :: List Int
list = Cons ( -3) ( Cons 14 ( Cons ( -6) ( Cons 7 ( Cons 1 Nil ))))


blist :: List Int
blist = Cons 1 ( Cons 1 ( Cons 0 ( Cons 0 Nil )))


--a)
filterList :: (a -> Bool) -> List a -> List a
filterList f Nil = Nil
filterList f (Cons a b) | (f a) == False = filterList f b
                        | (f a) == True = Cons a (filterList f b)



--b)
divisibleBy :: Int -> List Int -> List Int
divisibleBY x Nil = Nil
divisibleBy x (Cons a b) = filterList (\a -> (rem a x) == 0) (Cons a b)


--c)
foldList :: (a -> b -> b) -> b -> List a -> b
foldList f x Nil = x
foldList f x (Cons a b) = (f a) (foldList f x b)

plus :: Int -> Int -> Int
plus x y = x + y


--d)
listMaximum :: List Int -> Int
listMaximum Nil = minBound
listMaximum (Cons a b) = foldList (\x y -> if x > y then x else y) minBound (Cons a b)


--e)
mapList :: (a -> b) -> List a -> List b
mapList f xs = foldList (\y ys -> Cons(f y) ys) Nil xs


--f)
zipLists :: (a -> b -> c) -> List a -> List b -> List c
zipLists f Nil a = Nil
zipLists f a Nil = Nil
zipLists f (Cons a b) (Cons c d) = Cons (f a c) (zipLists f b d)


--g)
skalarprodukt :: List Int -> List Int -> Int
skalarprodukt Nil a = 0
skalarprodukt a Nil = 0
skalarprodukt (Cons a b) (Cons c d) = foldList (\x y -> x + y) 0 (zipLists (\x y -> x * y)
(Cons a b) (Cons c d))
```

Maximilian Petri, 405602
Danje Petersen, 379748

Hausaufgabe 11

Aufgabe 8)

```haskell
import Data.List

strings :: Int -> [String]
strings 0 = [""]
strings n = concat (map (\x -> map (\tail -> x:tail) tails) ['a'..'z'])
    where tails = strings (n-1)

divisors :: Int -> [Int]
divisors x = filter (\y -> rem x y == 0) [1..div x 2]

-- a)

palindromes :: [String]
palindromes = palindromesL 1

palindromesL :: Int -> [String]
palindromesL n = (map(\x -> x++(reverse x)) (strings n))++(palindromesL (n+1))

-- b)

perfects :: [Int]
perfects = filter isPerfect [2..]

isPerfect :: Int -> Bool
isPerfect x = x == sum (divisors x)

-- c)

semiPerfects :: [Int]
semiPerfects = filter isSemiPerfect [2..]

isSemiPerfect :: Int -> Bool
isSemiPerfect x = any (sumEquals x) (subsequences (divisors x))

sumEquals :: Int -> [Int] -> Bool
sumEquals _ [] = False
sumEquals n xs = n == sum xs

-- d)

fibs :: [Int]
fibs = fibInit 0 1

fibInit :: Int ->  Int -> [Int]
fibInit f0 f1 = f0 : f1 : next : fibInit (next + f1) (next + next + f1)
    where next = (f0 + f1)
```