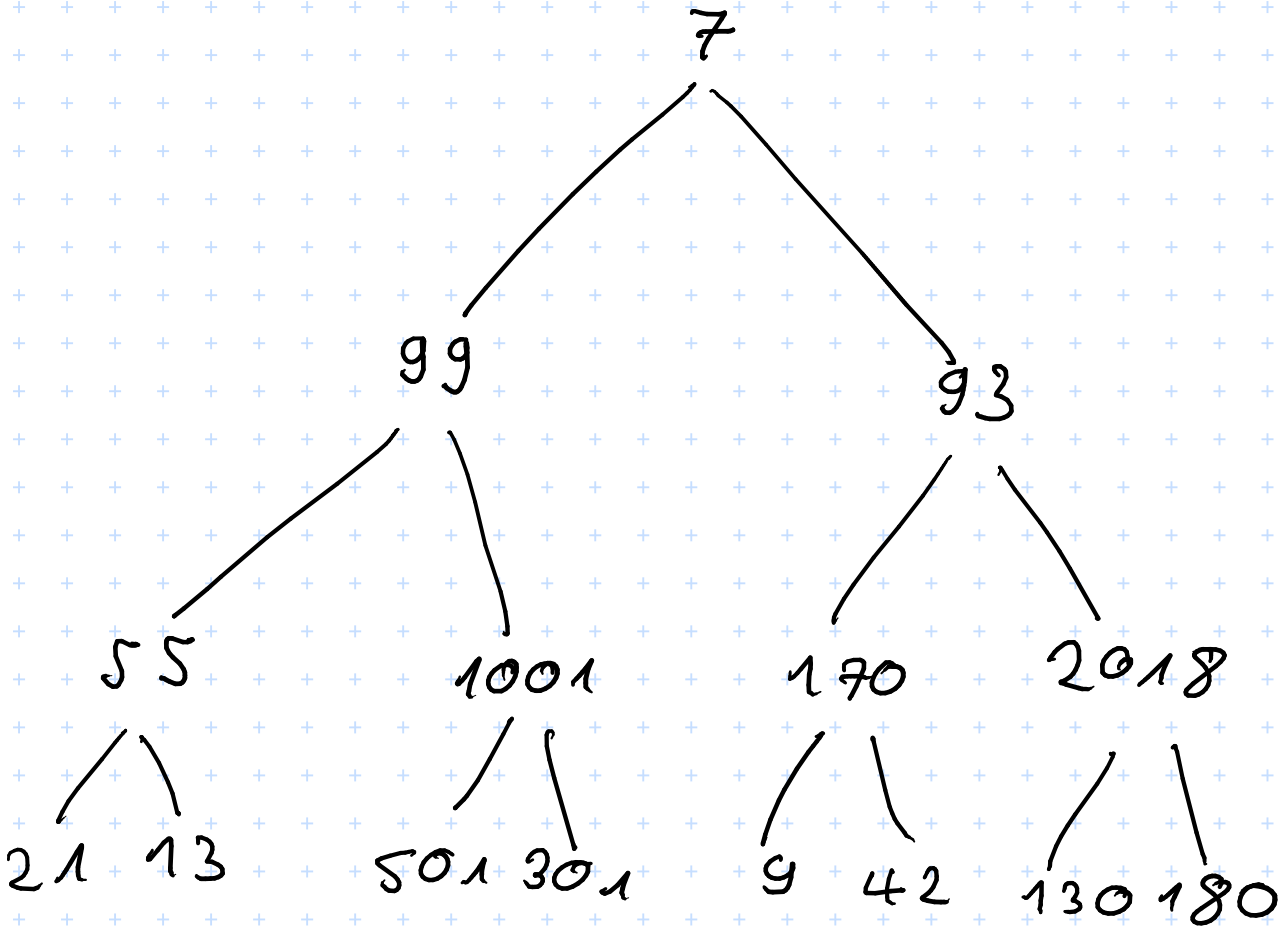
Lars Gröber - 5692604 - Gruppe 7

1) a)



```haskell
--
================================================================================
-- Grundlagen der Programmierung 2
-- Aufgabenblatt 5
--
================================================================================

    module Blatt5_LarsGroeber where
    import Data.Char
    import Data.List

    --
================================================================================
    -- =
    -- = Aufgabe 1
    -- =

    data BBaum a = BBlatt a | BKnoten a (BBaum a) (BBaum a)
      deriving(Eq,Show)

    data NBaum a = NBlatt a | NKnoten a [NBaum a]
      deriving(Eq,Show)

    --
    -- b)
    --
    musik = NKnoten "Musik"
        [
            NKnoten "Techno"
            [
                NBlatt "Re-Flex - Ubap",
                NBlatt "Dominion - Salted Popcord",
                NBlatt "XS Project - The Real Bass"
            ],
            NKnoten "Rock"
            [
                NBlatt "CSS - I Fly",
                NBlatt "The Offspring - Pretty Fly",
                NBlatt "Die Ärzte - Deine Schuld"
            ],
            NKnoten "Klassik"
            [
                NBlatt "Beethoven - Mondschein Senate",
                NBlatt "Mozart - Zauberflöte",
                NBlatt "Tchaikovsky - Schwanensee"
            ]
        ]

    --
    -- c)
    --
    getValueB :: BBaum a -> a
    -- returns the marking of a leaf
    getValueB (BBlatt a) = a
    getValueB (BKnoten a _ _) = a

    allaggr :: BBaum Int -> Bool
    -- Funktion, die testet, ob in einem Baum die Knoten- und
Blattmarkierungen korrekt sind
    allaggr (BBlatt _) = True
```

```haskell
    allaggr (BKnoten sum b1 b2) = (sum == getValueB b1 + getValueB b2) && allaggr b1 && allaggr b2

    {-
    Testfälle:
    testBaum = BKnoten 5 (BBlatt 2) (BBlatt 3)
    testBaum2 = BKnoten 4 (BBlatt 2) (BBlatt 3)
    testBaum3 = BKnoten 4 (BKnoten 3 (BBlatt 2) (BBlatt 1)) (BBlatt 1)
    testBaum4 = BKnoten 5 (BKnoten 3 (BBlatt 2) (BBlatt 1)) (BBlatt 1)

    allaggr testBaum `shouldBe` True
    allaggr testBaum2 `shouldBe` False
    allaggr testBaum3 `shouldBe` True
    allaggr testBaum4 `shouldBe` False
    -}

    --
    -- d)
    --
    toNTree :: BBaum a -> NBaum a
    -- converts a given BBaum to a NBaum
    toNTree (BBlatt a) = NBlatt a
    toNTree (BKnoten a b1 b2) = NKnoten a [toNTree b1, toNTree b2]

    listToNTree :: [NBaum (BBaum a)] -> [NBaum (NBaum a)]
    -- converts a list of NBaum (BBaum a) to NBaum (NBaum a)
    listToNTree [] = []
    listToNTree (NKnoten a list : xs) = NKnoten (toNTree a) (listToNTree list) : listToNTree xs
    listToNTree (NBlatt a : xs) = NBlatt (toNTree a) : listToNTree xs

    btreestontrees :: NBaum (BBaum a) -> NBaum (NBaum a)
    -- Funktion, die einen NBaum mit BBaum Markierungen entgegen nimmt und ihn in einen
    -- NBaum mit NBaum Markierungen konvertiert.
    btreestontrees (NKnoten bbaum list) = NKnoten (toNTree bbaum) (listToNTree list)

    {-
    Testfälle:
    testBaum2_1 = NKnoten (BKnoten 1 (BBlatt 1) (BBlatt 1)) []
    testBaum2_1_sol = NKnoten (NKnoten 1 [NBlatt 1, NBlatt 1]) []

    testBaum2_2 = NKnoten (BKnoten 1 (BBlatt 1) (BBlatt 1)) [testBaum2_1, testBaum2_1]
    testBaum2_2_sol = NKnoten (NKnoten 1 [NBlatt 1, NBlatt 1]) [testBaum2_1_sol, testBaum2_1_sol]

    btreestontrees testBaum2_1 `shouldBe` testBaum2_1_sol
    btreestontrees testBaum2_2 `shouldBe` testBaum2_2_sol
    -}

    --
================================================================================
    -- =
    -- = Aufgabe 2
    -- =

    type Note = (Int, Int, Int, Int)
    -- (Oktave,Tonh"o"he in der Oktave,L"ange,Lautst"arke)
```

```haskell
    type Noten = [(Int,Int,Int,Int)]
    -- Baum-Datenstruktur f"ur Melodien und Basslines.
    data Baum = Blatt Noten | Knoten Noten [Baum]
      deriving(Eq,Show)
    -- Datenstruktur f"ur die gesamten Entw"urfe.
    data Entwuerfe = Entwuerfe Baum Baum Int
      deriving(Eq,Show)

    --
    -- a)
    --
    anzahlNotenMB :: Entwuerfe -> Int
    -- Funktion, die einen Entwurf entgegen nimmt und die Anzahl der Noten
    -- im Melodiebaum zurückgibt.
    anzahlNotenMB (Entwuerfe m _ _) = anzahlNotenMBIt m

    anzahlNotenMBIt :: Baum -> Int
    -- returns the number of notes for a given Baum
    anzahlNotenMBIt (Blatt a) = length a
    anzahlNotenMBIt (Knoten a list) = length a + anzahlNotenMBList list

    anzahlNotenMBList :: [Baum] -> Int
    -- returns the number of notes of a list of Baum
    anzahlNotenMBList list = sum $ map anzahlNotenMBIt list

    {-
    Testfälle:
    note = (1,2,3,4)
    mBaum1 = Knoten [note, note, note] []
    mBaum2 = Knoten [note, note, note] [Knoten [note, note] [Blatt [note],
Blatt [note]]]

    anzahlNotenMB (Entwuerfe mBaum1 mBaum1 1) `shouldBe` 3
    anzahlNotenMB (Entwuerfe mBaum2 mBaum1 1) `shouldBe` 7
    -}

    --
    -- b)
    --
    transposeNote :: Int -> Note -> Note
    -- transposes a single note
    transposeNote value (a,b,c,d) = (a + ((tb - 1) `div` 12), (12 + tb - 1)
`mod` 12 + 1, c, d)
      where tb = b + value

    transposeNoten :: Int -> Noten -> Noten
    -- transposes a list of notes
    transposeNoten value = map (transposeNote value)

    transponiere :: Entwuerfe -> Int -> Entwuerfe
    -- Funktion, die einen Entwurf um einen gegebenen Wert transponiert
    transponiere (Entwuerfe m b t) value = Entwuerfe (transponiereIt value m)
(transponiereIt value b) t

    transponiereIt :: Int -> Baum -> Baum
    -- transposes a Baum
    transponiereIt value (Blatt a) = Blatt (transposeNoten value a)
    transponiereIt value (Knoten a list) = Knoten (transposeNoten value a)
(map (transponiereIt value) list)

```

```
166     {-
167     Testfälle:
168     note = (1,2,3,4)
169     mBaum1 = Knoten [note, note, note] []
170     mBaum2 = Knoten [note, note, note] [Knoten [note, note] [Blatt [note],
    Blatt [note]]]
171     noteT = (1,4,3,4)
172     mBaum1T = Knoten [noteT, noteT, noteT] []
173     mBaum2T = Knoten [noteT, noteT, noteT] [Knoten [noteT, noteT] [Blatt
    [noteT], Blatt [noteT]]]
174
175     transposeNote 3 (1,1,1,1)       `shouldBe` (1,4,1,1)
176     transposeNote 3 (1,9,1,1)       `shouldBe` (1,12,1,1)
177     transposeNote 3 (1,10,1,1)      `shouldBe` (2,1,1,1)
178     transposeNote (-3) (1,10,1,1)   `shouldBe` (1,7,1,1)
179     transposeNote (-3) (1,3,1,1)    `shouldBe` (0,12,1,1)
180
181     transponiere (Entwuerfe mBaum1 mBaum1 1) 2 `shouldBe` (Entwuerfe mBaum1T
    mBaum1T 1)
182     transponiere (Entwuerfe mBaum2 mBaum1 1) 2 `shouldBe` (Entwuerfe mBaum2T
    mBaum1T 1)
183     -}
```

3)

a)

map :: (a -> b) -> [a] -> [b]

iterate :: (c -> c) -> c -> [c]

gesucht: map iterate, γ ([a] -> [b])

G	E
$\emptyset$	$\{a \to b \doteq (c \to c) \to c \to [c]\}$
$\emptyset$	$\{a \doteq (c \to c),\ b \doteq c \to [c]\}$
$\{a \mapsto (c \to c),$ $b \mapsto c \to [c]\}$	$\emptyset$

$\Rightarrow$ map iterate :: $[c \to c] \to [c \to [c]]$ //

b)

gesucht: iterate map, γ (c -> [c])

G	E
$\emptyset$	$\{c \to c \doteq (a \to b) \to [a] \to [b]\}$
$\emptyset$	$\{c \to (a \to b),\ c \to [a] \to [b]\}$
$\{c \mapsto (a \to b)\}$	$\{a \to b \doteq [a] \to [b]\}$
"	$\{a \doteq [a],\ b \doteq [b]\}$
	Occurs check

c)

iterate :: $(c \to c) \to c \to [c]$

flip :: $(d \to e \to f) \to (e \to d \to f)$

const :: $g \to h \to g$

iterate flip const

gesucht: $\tau([c])$

G	E
$\emptyset$	$\{c \to c \doteq (d \to e \to f) \to (e \to d \to f),\ c \doteq g \to h \to g\}$
$\emptyset$	$\{c \doteq (d \to e \to f),\ c \doteq (e \to d \to f),\ c \doteq g \to h \to g\}$
$\{c \mapsto (d \to e \to f)\}$	$\{d \to e \to f \doteq e \to d \to f,\ d \to e \to f \doteq g \to h \to g\}$
"	$\{d \doteq e, e \doteq d, f \doteq f, d \doteq g, e \doteq h, f \doteq g\}$
$\{c \mapsto (e \to e \to f), d \mapsto e\}$	$\{e \doteq e, e \doteq g, e \doteq h, f \doteq g\}$
$\{c \mapsto (g \to g \to f), d \mapsto g, e \mapsto g\}$	$\{g \doteq h, f \doteq g\}$
$c \mapsto (h \to h \to f), d \mapsto h, e \mapsto h, g \mapsto h\}$	$\{f \doteq h\}$
$\{c \mapsto (h \to h \to h), d \to h, e \mapsto h, g \mapsto h, f \mapsto h\}$	$\emptyset$

$\Rightarrow$ iterate flip const :: $[h \to h \to h]$ //