Laborator 8
Arbori binari. Supraincarcarea operatorilor

1. Creati tipul Tree din 2.2.1 si testati functia "fringe".

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)

fringe :: Tree a -> [a]
fringe (Leaf x) = [x]
fringe (Branch left right) = fringe left ++ fringe right

main = print (fringe (Branch (Branch (Leaf 1) (Leaf 3)) (Branch (Leaf 2) (Leaf 4))))
```

2. Scrieti o functie care cauta o valoare intr-un arbore de tip Tree.

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)

mySearch :: Tree Integer -> Integer -> Bool
mySearch (Leaf x) e = x == e
mySearch (Branch left right) e = mySearch left e || mySearch right e

main = print (mySearch (Branch (Branch (Leaf 1) (Leaf 3)) (Branch (Leaf 2) (Leaf 4)))
3)
```

3. Creati tipul Tree1 pentru arbori care au valori in toate nodurile, nu doar in frunze. Scrieti o functie care reintoarce parcurgerea in inordine a unui arbore de tip Tree1.

```
data Tree1 a = Leaf a | Branch a (Tree1 a) (Tree1 a)

srd1 :: Tree1 a -> [a]
srd1 (Leaf x) = [x]
srd1 (Branch x left right) = srd1 left ++ [x] ++ srd1 right

main = print (srd1 (Branch 1 (Branch 2 (Leaf 4) (Leaf 5)) (Leaf 3)))
```

4+5. Creati tipul Tree2 care accepta arbori de tip Tree1 si arborele vid. Scrieti o functie care inlocuieste toate aparitiile unei valori x cu o valoare y intr-un arbore de tip Tree2.

```
-- am eliminat constructorul Leaf
-- folosesc Empty si pentru identificarea frunzelor
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a)
```

```
srd2 :: Tree2 a -> [a]
srd2 Empty = []
srd2 (Branch x left right) = srd2 left ++ [x] ++ srd2 right

myReplace :: Tree2 Integer -> Integer -> Integer -> Tree2 Integer
myReplace Empty e1 e2 = Empty
myReplace (Branch x left right) e1 e2 = if x == e1 then (Branch e2 (myReplace left e1
e2) (myReplace right e1 e2))
                                        else (Branch x (myReplace left e1 e2) (myReplace right
e1 e2))

main = print (srd2 (myReplace (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5
Empty Empty)) (Branch 2 Empty Empty)) 2 7))
```

6. Scrieti o functie care sa oglindeasca un arbore de tip Tree2.

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a)

srd2 :: Tree2 a -> [a]
srd2 Empty = []
srd2 (Branch x left right) = srd2 left ++ [x] ++ srd2 right

myMirror :: Tree2 a -> Tree2 a
myMirror Empty = Empty
myMirror (Branch x left right) = (Branch x (myMirror right) (myMirror left))

main = print (srd2 (myMirror (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5
Empty Empty)) (Branch 3 Empty Empty))))
```

7. Scrieti o functie care gaseste subarborele de suma maxima dintr-un arbore de tip Tree2.

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a)

srd2 :: Tree2 a -> [a]
srd2 Empty = []
srd2 (Branch x left right) = srd2 left ++ [x] ++ srd2 right

sumArb :: Tree2 Integer -> Integer
sumArb Empty = 0
sumArb (Branch x left right) = x + sumArb left + sumArb right

arbList :: Tree2 Integer -> [(Tree2 Integer, Integer)]
arbList Empty = []
arbList (Branch x Empty Empty) = [(Branch x Empty Empty,x)]
```

```
arbList (Branch x left right) = [(Branch x left right,sumArb (Branch x left right))] ++
arbList left ++ arbList right

arbSumMax :: [(Tree2 Integer, Integer)] -> (Tree2 Integer, Integer) -> Tree2 Integer
arbSumMax [] (t,s) = t
arbSumMax ((tx,sx):lst) (t,s) = if sx > s then arbSumMax lst (tx,sx) else arbSumMax lst
(t,s)

solutie :: Tree2 Integer -> Tree2 Integer
solutie t = arbSumMax (arbList t) (t,sumArb t)

main = print (srd2 (solutie (Branch 100 (Branch (-20) (Branch 4 Empty Empty) (Branch
5 Empty Empty)) (Branch 30 Empty Empty))))
```

8. Supraincarcati operatorul de egalitate "==" astfel incat sa functioneze si pentru arbori de tip Tree2.

Metoda 1

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a) deriving Eq

t1 = (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty)) (Branch 3
Empty Empty))
t2 = (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty)) (Branch 3
Empty Empty))

main = print (t1 == t2)
```

Metoda 2

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a)

instance (Eq a) => Eq (Tree2 a) where
  Empty == Empty = True
  (Branch x1 left1 right1) == (Branch x2 left2 right2) = (x1 == x2) && (left1 == left2)
&& (right1 == right2)
  _ == _ = False

t1 = (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty)) (Branch 3
Empty Empty))
t2 = (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty)) (Branch 3
Empty Empty))

main = print (t1 == t2)
```

9. Supraincarcati Show (utilizat de print) pentru a afisa arbori binari de tip Tree2.

Metoda 1

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a) deriving Show

main = print (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty))
(Branch 3 Empty Empty))
```

Metoda 2

```
data Tree2 a = Empty | Branch a (Tree2 a) (Tree2 a)

instance (Show a) => Show (Tree2 a) where
  show Empty = "Empty"
  show (Branch x left right) = "(Branch" ++ " " ++ show x ++ " " ++ (show left) ++ " " ++
(show right) ++ ")"

main = print (Branch 1 (Branch 2 (Branch 4 Empty Empty) (Branch 5 Empty Empty))
(Branch 3 Empty Empty))
```