Laboratorul 7: ADT

Expresii și arbori

Se dau următoarele tipuri de date reprezentând expresii și arbori de expresii:

```
data Expr = Const Int -- integer constant
          | Expr :+: Expr -- addition
          | Expr :*: Expr -- multiplication
           deriving Eq

data Operation = Add | Mult deriving (Eq, Show)

data Tree = Lf Int -- leaf
          | Node Operation Tree Tree -- branch
           deriving (Eq, Show)
```

Pentru a afișa mai simplu expresiile, putem instanția clasa `Show` pentru tipul de date `Expr` astfel:

```
instance Show Expr where
  show (Const x) = show x
  show (e1 :+: e2) = "(" ++ show e1 ++ " + "++ show e2 ++ ")"
  show (e1 :*: e2) = "(" ++ show e1 ++ " * "++ show e2 ++ ")"
```

1. Scrieți o funcție `evalExp :: Expr -> Int` care evaluează o expresie determinând valoarea acesteia.

```
evalExp :: Expr -> Int
evalExp = undefined
```

Exemplu:

```
exp1 = ((Const 2 :*: Const 3) :+: (Const 0 :*: Const 5))
exp2 = (Const 2 :*: (Const 3 :+: Const 4))
exp3 = (Const 4 :+: (Const 3 :*: Const 3))
exp4 = (((Const 1 :*: Const 2) :*: (Const 3 :+: Const 1)) :*: Const 2)
test11 = evalExp exp1 == 6
test12 = evalExp exp2 == 14
test13 = evalExp exp3 == 13
test14 = evalExp exp4 == 16
```

2. Scrieți o funcție `evalArb :: Tree -> Int` care evaluează o expresie modelată sub formă de arbore, determinând valoarea acesteia.

```
evalArb :: Tree -> Int
evalArb = undefined

arb1 = Node Add (Node Mult (Lf 2) (Lf 3)) (Node Mult (Lf 0)(Lf 5))
arb2 = Node Mult (Lf 2) (Node Add (Lf 3)(Lf 4))
arb3 = Node Add (Lf 4) (Node Mult (Lf 3)(Lf 3))
arb4 = Node Mult (Node Mult (Node Mult (Lf 1) (Lf 2)) (Node Add (Lf 3)(Lf 1))) (Lf 2)

test21 = evalArb arb1 == 6
test22 = evalArb arb2 == 14
test23 = evalArb arb3 == 13
test24 = evalArb arb4 == 16
```

3. Scrieți o funcție `expToArb :: Expr -> Tree` care transformă o expresie în arborele corespunzător.

```
expToArb :: Expr -> Tree
expToArb = undefined
```

Arbori binari de căutare

Fie tipul arborilor binari de căutare (ne-echilibrați) cu noduri constând în perechi chei-valoare cu chei numere întregi:

```
data IntSearchTree value
  = Empty
  | BNode
      (IntSearchTree value)      -- elemente cu cheia mai mica
      Int                        -- cheia elementului
      (Maybe value)              -- valoarea elementului
      (IntSearchTree value)      -- elemente cu cheia mai mare
```

Observați că tipul valorilor este `Maybe value`. Alegerea a fost făcută pentru a reduce timpul operației de ștergere prin simpla marcare a unui nod ca fiind șters. Un nod șters va avea valoarea `Nothing`.

4. Scrieți o funcție `lookup'` de căutare a unui element într-un arbore.

```
lookup' :: Int -> IntSearchTree value -> Maybe value
lookup' = undefined
```

5. Scrieți o funcție care întoarce lista cheilor nodurilor dintr-un arbore de căutare.

```
keys ::  IntSearchTree value -> [Int]
keys = undefined
```

6. Scrieți o funcție care întoarce lista valorilor nodurilor dintr-un arbore de căutare.

```
values :: IntSearchTree value -> [value]
values = undefined
```

7. Scrieți o funcție de adăugare a unui element într-un arbore de căutare.

```
insert :: Int -> value -> IntSearchTree value -> IntSearchTree value
insert = undefined
```

8. Scrieți o funcție care șterge (marchează ca șters) un element dintr-un arbore de căutare.

```
delete :: Int -> IntSearchTree value -> IntSearchTree value
delete = undefined
```

9. Scrieți o funcție care întoarce lista elementelor dintr-un arbore de căutare. Hint: atenție la `Maybe`!

```
toList :: IntSearchTree value -> [(Int, value)]
toList = undefined
```

10. Scrieți o funcție care să construiască un arbore dintr-o listă de perechi cheie-valoare.

```
fromList :: [(Int, value)] -> IntSearchTree value
fromList = undefined
```

11. Scrieți o funcție care să producă o reprezentare liniară (șir de caractere) a structurii arborescente de chei (ignorând valorile). De exemplu, arborele cu rădăcina cu cheia 2, copilul stâng cu cheia 1 și copilul drept cu cheia 3 ar putea fi reprezentat ca "(1) 2 (3)". Puteți alege și alte reprezentări.

```
printTree :: IntSearchTree value -> String
printTree = undefined
```

Extra

12. Scrieți o funcție care primește ca parametru un arbore binar de căutare și întoarce arborele echilibrat.

```
balance :: IntSearchTree value -> IntSearchTree value
balance = undefined
```