

Laboratorul 9: ADT

1. 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)
```

1.1. Să se instanțieze clasa `Show` pentru tipul de date `Expr`, astfel încât să se afișeze mai simplu expresiile. 1.2. Să se scrie 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
```

1.3. Să se scrie o funcție `evalArb :: Tree -> Int` care evaluează o expresie determinând valoarea acesteia.

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

1.4. Să se scrie o funcție `expToArb :: Expr -> Tree` care transformă o expresie în arborele corespunzător.

```

expToArb :: Expr -> Tree
expToArb = 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

```

1.5. Să se instanțieze clasa `MySmallCheck` (Clasa tipurilor “mici” - *cursul 8*) pentru tipul de date `Expr`, lista de valori conținând câteva expresii definite de voi.

```

class MySmallCheck a where
    smallValues :: [a]
    smallCheck :: ( a -> Bool ) -> Bool
    smallCheck prop = and [ prop x | x <- smallValues ]

```

1.6. Să se scrie un predicat care verifică faptul că evaluarea unei expresii este egală cu evaluarea arborelui asociat expresiei. Folosind funcția `smallCheck` să se verifice ca predicatul este adevărat pentru toate valorile `smallValues`.

```

checkExp :: Expr -> Bool
checkExp = undefined

```

2. Clasa Collection

În acest exercitiu vom exersa manipularea listelor și tipurilor de date prin implementarea a câtorva colecții de tip tabelă asociativă cheie-valoare.

Aceste colecții vor trebui să aibă următoarele facilități

- crearea unei colecții vide
- crearea unei colecții cu un element
- adăugarea/actualizarea unui element într-o colecție
- căutarea unui element într-o colecție
- ștergerea (marcarea ca șters a) unui element dintr-o colecție
- obținerea listei cheilor
- obținerea listei valorilor
- obținerea listei elementelor

```

import           Prelude hiding (lookup)
import qualified Data.List as List

class Collection c where
    empty :: c key value
    singleton :: key -> value -> c key value

```

```

insert
  :: Ord key
  => key -> value -> c key value -> c key value
lookup :: Ord key => key -> c key value -> Maybe value
delete :: Ord key => key -> c key value -> c key value
keys   :: c key value -> [key]
values :: c key value -> [value]
toList :: c key value -> [(key, value)]
fromList :: Ord key => [(key,value)] -> c key value

```

2.1. Adaugati definitii implicite (in functie de functiile celelalte) pentru

- a. keys
- b. values
- c. fromList

2.2. Fie tipul listelor de perechi de forma cheie-valoare:

```

newtype PairList k v
  = PairList { getPairList :: [(k, v)] }

```

Faceti PairList instanta a clasei Collection.

2.3. Fie tipul arborilor binari de cautare (ne-echilibrati):

```

data SearchTree key value
  = Empty
  | Node
    (SearchTree key value) -- elemente cu cheia mai mica
    key                    -- cheia elementului
    (Maybe value)          -- valoarea elementului
    (SearchTree key value) -- elemente cu cheia mai mare

```

Observati ca tipul valorilor este `Maybe value`. Acest lucru se face pentru a reduce timpul operatiei de stergere prin simpla marcare a unui nod ca fiind sters. Un nod sters va avea valoarea `Nothing`.

Faceti `SearchTree` instanta a clasei `Collection`.