Laboratorul 8: ADT. Clase de tipuri

Clasa Collection

În acest exercițiu vom exersa manipularea listelor și a tipurilor de date prin definirea câtorva colecții de tip tabelă asociativă cheie-valoare. Pentru aceste colecții vom implementa:

- 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

```
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
```

1. Adăugați definiții implicite (folosind celelalte funcții din clasă) pentru `keys`, `values` și `fromList`.

2. Fie tipul listelor de perechi cheie-valoare:

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

Faceți `PairList` instanță a clasei `Collection`.

3. Amintiți-vă exercițiul din laboratorul trecut în care ați definit tipul arborilor de căutare cu noduri constând în perechi chei-valoare cu chei numere întregi. Vom generaliza acest tip definind arbori binari de căutare (ne-echilibrați) cu chei de tip oarecare:

```
data SearchTree key value
  = Empty
  | BNode
      (SearchTree key value) -- elemente cu cheia mai mica
      key                    -- cheia elementului
      (Maybe value)          -- valoarea elementului
      (SearchTree key 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`.

Faceți `SearchTree` instanță a clasei `Collection`.

Puncte puncte

Se dau două tipuri de date ce reprezintă puncte cu număr variabil de coordonate întregi și arbori cu informație salvată în frunze, și o clasă de tipuri `ToFromArb`.

```
data Punct = Pt [Int]
```

```
data Arb = Vid | F Int | N Arb Arb
          deriving Show

class ToFromArb a where
      toArb :: a -> Arb
      fromArb :: Arb -> a
```

4. Scrieți o instanță a clasei `Show` pentru tipul de date `Punct`, astfel încât lista coordonatelor să fie afișată ca tuplu.

```
-- Pt [1,2,3]
-- (1, 2, 3)

-- Pt []
-- ()
```

5. Scrieți o instanță a clasei `ToFromArb` pentru tipul de date `Punct` astfel încât lista coordonatelor punctului să coincidă cu frontiera arborelui.

```
-- toArb (Pt [1,2,3])
-- N (F 1) (N (F 2) (N (F 3) Vid))
-- fromArb $ N (F 1) (N (F 2) (N (F 3) Vid)) :: Punct
--  (1,2,3)
```

Figuri geometrice

Se dau următorul tip de date reprezentând figuri geometrice

```
data Geo a = Square a | Rectangle a a | Circle a
    deriving Show
```

și clasa `GeoOps` în care se definesc operațiile `perimeter` și `area`:

```
class GeoOps g where
  perimeter :: (Floating a) => g a -> a
  area :: (Floating a) =>  g a -> a
```

6. Instanțiați clasa `GeoOps` pentru tipul de date `Geo`. Hint: pentru valoarea `pi` puteți folosi funcția cu același nume (`pi`).

```
-- ghci> pi
-- 3.141592653589793
```

7. Instanțiați clasa `Eq` pentru tipul de date `Geo`, astfel încât două figuri geometrice să fie egale dacă au perimetrul egal.