Laboratorul 6: Tipuri de date

Apples and Oranges

1. Vom începe prin a scrie câteva funcții definite folosind tipul de date `Fruct`:

```
data Fruct
  = Mar String Bool
  | Portocala String Int
```

O expresie de tipul `Fruct` este fie un `Mar String Bool`, fie o `Portocala String Int`. Vom folosi un `String` pentru a indica soiul de mere sau de portocale, un `Bool` pentru a indica dacă mărul are viermi, și un `Int` pentru a exprima numărul de felii dintr-o portocală. De exemplu:

```
ionatanFaraVierme = Mar "Ionatan" False
goldenCuVierme = Mar "Golden Delicious" True
portocalaSicilia10 = Portocala "Sanguinello" 10
cosFructe = [Mar "Ionatan" False,
                Portocala "Sanguinello" 10,
                Portocala "Valencia" 22,
                Mar "Golden Delicious" True,
                Portocala "Sanguinello" 15,
                Portocala "Moro" 12,
                Portocala "Tarocco" 3,
                Portocala "Moro" 12,
                Portocala "Valencia" 2,
                Mar "Golden Delicious" False,
                Mar "Golden" False,
                Mar "Golden" True]
```

a) Scrieți un predicat

```
ePortocalaDeSicilia :: Fruct -> Bool
ePortocalaDeSicilia = undefined
```

care verifică dacă un fruct este o portocală de Sicilia. Soiurile de portocale din Sicilia sunt `Tarocco`, `Moro` și `Sanguinello`. De exemplu,

```
test_ePortocalaDeSicilia1 =
    ePortocalaDeSicilia (Portocala "Moro" 12) == True
```

```
test_ePortocalaDeSicilia2 =
    ePortocalaDeSicilia (Mar "Ionatan" True) == False
```

b) Scrieți o funcție

```
nrFeliiSicilia :: [Fruct] -> Int
nrFeliiSicilia = undefined

test_nrFeliiSicilia = nrFeliiSicilia listaFructe == 52
```

care calculează numărul total de felii ale portocalelor de Sicilia dintr-o listă de fructe.

c) Scrieți o funcție

```
nrMereViermi :: [Fruct] -> Int
nrMereViermi = undefined

test_nrMereViermi = nrMereViermi listaFructe == 2
```

care calculează numărul de mere care au viermi dintr-o listă de fructe.

Paw Patrol

2. Se dă tipul de date `Animal`.

```
type NumeA = String
type Rasa = String
data Animal = Pisica NumeA | Caine NumeA Rasa
    deriving Show
```

a) Scrieți o funcție

```
vorbeste :: Animal -> String
vorbeste = undefined
```

care întoarce `"Meow!"` pentru pisică și `"Woof!"` pentru câine.

b) Reamintiți-vă tipul de date predefinit `Maybe`.

```
data Maybe a = Nothing | Just a
```

Scrieți o funcție

```
rasa :: Animal -> Maybe String
rasa = undefined
```

care întoarce rasa unui câine dat ca parametru sau `Nothing` dacă parametrul este o pisică.

Matrix Resurrections

3. Se dau următoarele tipuri de date ce reprezintă matrici cu linii de lungimi diferite:

```
data Linie = L [Int]
     deriving Show
data Matrice = M [Linie]
   deriving Show
```

a) Scrieți o funcție care verifică dacă suma elementelor de pe fiecare linie este egală cu o valoare dată `n`. Rezolvați cerința folosind `foldr`.

```
verifica :: Matrice -> Int -> Bool
verifica = undefined

test_veri1 = verifica (M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 10 == False

test_verif2 = verifica (M[L[2,20,3], L[4,21], L[2,3,6,8,6], L[8,5,3,9]]) 25 == True
```

b) Scrieți o funcție `doarPozN` care are ca parametri un element de tip `Matrice` și un număr întreg `n`, și care verifică dacă toate liniile de lungime `n` din matrice au numai elemente strict pozitive.

```
doarPozN :: Matrice -> Int -> Bool
doarPozN = undefined

testPoz1 = doarPozN (M [L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3 == True

testPoz2 = doarPozN (M [L[1,2,-3], L[4,5], L[2,3,6,8], L[8,5,3]]) 3 == False
```

c) Definiți predicatul `corect` care verifică dacă toate liniile dintr-o matrice au aceeași lungime.

```
corect :: Matrice -> Bool
corect = undefined

testcorect1 = corect (M[L[1,2,3], L[4,5], L[2,3,6,8], L[8,5,3]]) == False
testcorect2 = corect (M[L[1,2,3], L[4,5,8], L[3,6,8], L[8,5,3]]) == True
```

Extra: Turtle!

4. Ajutați-l pe Donatello să găsească pizza folosind limbajul Turtle! Limbajul constă din comenzi și acțiuni prin care o țestoasă se poate mișca în plan.

a) Definiți un tip de date `Turtle` care să codifice poziția și orientarea țestoasei. Poziția corespunde coordonatelor carteziene în plan, iar orientarea corespunde punctelor cardinale.

b) Definiți un tip de date `Action` corespunzător următoarelor două acțiuni:

- `step`– pentru deplasarea țestoasei cu o poziție conform orientării curente
- `turn` – pentru schimbarea orientării cu 45 de grade în sensul acelor de ceasornic.

c) Definiți un tip de date `Command` cu următorii constructori:

- `do` – pentru executarea unei acțiuni
- `repeat` – pentru repetarea unei acțiuni de un număr dat de ori.

d) Scrieți o funcție `getPizza` care primește drept argumente o țestoasă și o listă de comenzi și returnează poziția țestoasei în urma executării comenzilor.

e) Extindeți tipul de date `Command` cu constructorul `wait` care corespunde comenzii vide. Extindeți tipul de date `Action` cu un constructor `seq` care are drept argumente două comenzi `c1` și `c2`. Modificați funcția `getPizza` pentru a folosi noii constructori:

- `wait` nu schimbă nici poziția, nici orientarea țestoasei
- `seq c1 c2` corespunde schimbării poziției sau orientării țestoasei mai întâi conform comenzii `c1`, iar apoi conform comenzii `c2`.

f) Folosiți `fold` pentru a agrega o listă de comenzi într-o singură comandă echivalentă.