Laboratorul 3: Liste

Recursivitate pe liste

1) Să se scrie o funcție `nrVocale` care pentru o listă de șiruri de caractere, calculează numărul total de vocale ce apar în cuvintele palindrom. Pentru a verifica dacă un șir e palindrom, puteți folosi funcția `reverse`, iar pentru a căuta un element într-o listă puteți folosi funcția `elem`. Puteți defini oricâte funcții auxiliare.

```
nrVocale :: [String] -> Int
nrVocale = undefined
-- nrVocale ["sos", "civic", "palton", "desen", "aerisirea"] = 9
```

2) Să se scrie o funcție care primește ca parametru un număr și o listă de întregi, și adaugă elementul dat după fiecare element par din listă. Să se scrie și prototipul funcției.

```
-- f 3 [1,2,3,4,5,6] = [1,2,3,3,4,3,5,6,3]
```

Liste definite prin comprehensiune sau selecție

Haskell permite definirea unei liste prin selectarea și transformarea elementelor din alte liste sursă, folosind o sintaxă asemănătoare definirii mulțimilor matematice:

`[expresie | selectori, legari, filtrari]`

unde:

selectori una sau mai multe construcții de forma `pattern <- elista` (separate prin virgulă) unde `elista` este o expresie reprezentând o listă iar `pattern` este un șablon pentru elementele listei `elista`

legari zero sau mai multe expresii (separate prin virgulă) de forma `let pattern = expresie` folosind la legarea corespunzătoare a variabilelor din `pattern` cu valoarea `expresie`.

filtrari zero sau mai multe expresii de tip `Bool` (separate prin virgulă) folosite la eliminarea instanțelor selectate pentru care condiția e falsă

expresie expresie descriind elementele listei rezultat

Exemplu Iată cum arată o posibilă implementare a funcției `semiPare` folosind descrieri de liste:

```
semiPareComp :: [Int] -> [Int]
semiPareComp l = [ x `div` 2 | x <- l, even x ]
```

Exercitii

3) Să se scrie o funcție care are ca parametru un număr întreg și determină lista de divizori ai acestui număr. Să se scrie și prototipul funcției.

```
-- divizori 4 = [1,2,4]
```

4) Să se scrie o funcție care are ca parametru o listă de numere întregi și calculează lista listelor de divizori.

```
listadiv :: [Int] -> [[Int]]
listadiv = undefined
```

```
-- listadiv [1,4,6,8] = [[1],[1,2,4],[1,2,3,6],[1,2,4,8]]
```

5) Scrieți o funcție care date fiind limita inferioară și cea superioară (întregi) a unui interval închis și o listă de numere întregi, calculează lista numerelor din listă care aparțin intervalului. De exemplu:

```
-- inInterval 5 10 [1..15] == [5,6,7,8,9,10]
```

```
-- inInterval 5 10 [1,3,5,2,8,-1] = [5,8]
```

a) Folosiți doar recursie. Denumiți funcția `inIntervalRec`
b) Folosiți descrieri de liste. Denumiți funcția `inIntervalComp`

6) Scrieți o funcție care numără câte numere strict pozitive sunt într-o listă dată ca argument.

De exemplu:

```
-- pozitive [0,1,-3,-2,8,-1,6] == 3
```

a) Folosiți doar recursie. Denumiți funcția `pozitiveRec`
b) Folosiți descrieri de liste. Denumiți funcția `pozitiveComp`.
   - Nu puteți folosi recursie, dar veți avea nevoie de o funcție de agregare. (Consultați modulul Data.List). De ce nu e posibil să scriem `pozitiveComp` doar folosind descrieri de liste?

7) Scrieți o funcție care dată fiind o listă de numere calculează lista pozițiilor elementelor impare din lista originală. De exemplu:

```
-- pozitiiImpare [0,1,-3,-2,8,-1,6,1] == [1,2,5,7]
```

a) Folosiți doar recursie. Denumiți funcția `pozitiiImpareRec`.
   - Indicație: folosiți o funcție ajutătoare, cu un argument în plus reprezentând poziția curentă din listă.
b) Folosiți descrieri de liste. Denumiți funcția `pozitiiImpareComp`.

- Indicație: folosiți funcția `zip` pentru a asocia pozitii elementelor listei (puteți căuta exemplu în curs).

8) Scrieți o funcție care calculează produsul tuturor cifrelor care apar în șirul de caractere dat ca intrare. Dacă nu sunt cifre în șir, răspunsul funcției trebuie să fie `1` . De exemplu:

```
-- multDigits "The time is 4:25" == 40
-- multDigits "No digits here!" == 1
```

a) Folosiți doar recursie. Denumiți funcția `multDigitsRec`
b) Folosiți descrieri de liste. Denumiți funcția `multDigitsComp`
   - Indicație: Veți avea nevoie de funcția `isDigit` care verifică dacă un caracter e cifră și funcția `digitToInt` care transformă un caracter in cifră. Cele 2 funcții se află în pachetul `Data.Char`.