Laboratorul 3: Liste

Liste și recursivitate

1. Scrieți o funcție `nrVocale` care primește ca parametru o listă de șiruri de caractere și calculează numărul total de vocale din șirurile 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 funcții auxiliare.

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

2. Scrieți o funcție care primește ca parametri un număr și o listă de întregi și adaugă numărul 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 cu proprietăți caracteristice sau prin selecție

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

`[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` ce folosesc 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` din Laboratorul 2 folosind descrieri de liste:

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

Exerciții

3. Scrieți o funcție care determină lista divizorilor unui număr întreg primit ca parametru. Să se scrie și prototipul funcției.

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

4. Scrieți o funcție care primește ca parametru o listă de numere întregi și întoarce 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 primește ca parametri:

   - două numere întregi ce reprezintă limita inferioară și cea superioară a unui interval închis și
   - o listă de numere întregi

   și întoarce numerele din listă ce 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) Definiți funcția recursiv și denumiți-o `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) Definiți funcția recursiv și denumiți-o `pozitiveRec`.

   b) Folosiți descrieri de liste. Denumiți funcția `pozitiveComp`.

      Hint: Nu puteți folosi recursivitate. Veți avea nevoie de o funcție de agregare (consultați modulul `Data.List`). De ce nu e posibil să scriem `pozitiveComp` folosind doar descrieri de liste?

7. Scrieți o funcție care întoarce lista pozițiilor elementelor impare dintr-o listă de numere primită ca parmetru. De exemplu:

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

a) Definiți funcția recursiv și denumiți-o `pozitiiImpareRec`.

   Hint: folosiți o funcție ajutătoare, cu un parametru în plus reprezentând poziția curentă din listă.

b) Folosiți descrieri de liste. Denumiți funcția `pozitiiImpareComp`.

   Hint: folosiți funcția `zip` pentru a asocia poziții elementelor listei (puteți găsi un exemplu în curs).

8. Scrieți o funcție care calculează produsul tuturor cifrelor care apar într-un șir de caractere primit ca parametru. Dacă șirul nu conține cifre, funcția întoarce `1` . De exemplu:

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

a) Definiți funcția recursiv și denumiți-o `multDigitsRec`.

b) Folosiți descrieri de liste. Denumiți funcția `multDigitsComp`.

   Hint: Veți avea nevoie de funcția `isDigit` care verifică dacă un caracter e cifră și de funcția `digitToInt` care transformă un caracter în cifră. Cele 2 funcții se află în pachetul `Data.Char`.

Extra

9. Scrieți o funcție care primește ca argument o listă și întoarce toate permutările ei.
10. Scrieți o funcție care primește ca argument o listă și un număr întreg `k`, și întoarce toate combinările de `k` elemente din listă.
11. Scrieți o funcție care primește ca argument o listă și un număr întreg `k`, și întoarce toate aranjamentele de `k` elemente din listă.
12. Scrieți o funcție care primește ca argument un număr întreg ce reprezintă dimensiunea unei table de șah și un numar întreg ce reprezintă numărul de dame ce trebuie așezate pe tablă, și întoarce lista pozițiilor în care pot fi așezate damele fără să se atace.