Поправителен изпит по Функционално програмиране
Специалност „Информационни системи“, I курс, 16.08.2021 г.

Задача 1

Да се дефинира функция **`seriesSum :: Double -> Int -> Double`**, която има два параметъра – реално число **`x`** и цяло число **`n`** > 0. Функцията да пресмята следната сума:

$$\sum_{i=1}^{n} a_i\,(x), \text{където } a_i\,(x) = 1 + x^i + i^2.$$

Примери:

seriesSum `1 3` ⟶ `20.0`

seriesSum `2 4` ⟶ `64.0`

Задача 2

Да се дефинира функция **`kthNumber :: [Int] -> (Int -> Bool) -> (Int -> Int)`**, която приема списък **`xs`** от цели числа и функция предикат **`p`**. Функцията **`kthNumber`** трябва да върне функция с параметър естествено число **`k`** – такава, че оценката на израза **`((kthNumber xs p) k)`** е **`k`**-тото подред число в списъка **`xs`**, което удовлетворява предиката **`p`**. Ако такова число не съществува, да се връща грешката “No such number”.

Примери:

```
(kthNumber [-2, 3, 5, -4, -13, -15, 20, -21] (> 5)) 12
                                              ⟶ error “No such number”
(kthNumber [-2, 3, 5, -4, -13, -15, 20, -21] (>= 5)) 2 ⟶ 20
```

Задача 3

Нека са дефинирани:

```
type PersonID = Int
type Name = String
type City = String
type AccountID = Int
type Balance = Double
type Person = (PersonID, Name, City)
type Account = (AccountID, PersonID, Balance)
```

Да се дефинира функцията **getCriticalBalance :: ([Account], [Person]) -> (Person -> Bool) -> Balance -> [(PersonID, Balance)]**. Функцията получава като първи аргумент база от данни **database,** представена като двойка от списък от сметки и списък от хора. Вторият аргумент е предикат **`p`**, а третият е критична стойност **`s`** на баланс. Функцията **getCriticalBalance** трябва да връща списък от двойки от идентификатор на човек (PersonID) и сума на баланс по всички сметки за конкретния PersonID. Върнатият списък трябва да съдържа всички хора, които удовлетворяват предиката **p** и за които сумата на баланса по всички сметки е по-малка от **`s`**. Всяка сметка е вектор с три елемента: идентификатор на сметка, идентификатор на човек и баланс по сметката. Всеки човек

е представен като вектор от три елемента, съответно: идентификатор на човек, име и местоживеене.

Примери:

```
db = ([(1, 1, 10), (2, 1, 11), (3, 1, 12), (4, 2, 3), (5, 2, 1), (6,
3, 2), (7, 3, 3), (8, 4, 12)], [(1, "Ivan", "Varna"), (2, "Petar",
"Burgas"), (3, "Georgi", "Varna"), (4, "Yordan", "Plovdiv")])

fromVarna :: Person -> Bool
fromVarna (_, _, "Varna") = True
fromVarna _ = False

getCriticalBalance db fromVarna 10       ⟶ [(3,5.0)]
getCriticalBalance db (not . fromVarna) 15 ⟶ [(2,4.0),(4,12.0)]
```

Задача 4

Нека за представянето на двоично дърво от цели числа се използва алгебричен тип със следната дефиниция:

data BTree = Empty | Node Int BTree Btree

Да се дефинира функция **findNodes :: BTree -> [Int]**, която получава като единствен аргумент двоично дърво, в което стойността във всеки възел е уникална. Функцията трябва да върне сортиран във възходящ ред списък. Списъкът трябва да съдържа стойностите на тези възли, за които е в сила следното: възелът има наследници и стойността във възела е по-голяма от сумата на стойностите на преките наследници на този възел.

Примери:

```
t1 = Node 1 (Node 12 (Node 2 Empty Empty)
                     (Node 3 Empty Empty))
            (Node 20 (Node 17 (Node 13 Empty Empty)
                              Empty)
                     Empty)

findNodes t1 ⟶ [12, 17, 20]

t2 = Node 10 (Node 2 Empty
                     (Node 3 (Node 4 Empty Empty)
                             Empty))
             (Node 11 (Node 1 Empty Empty)
                      (Node 6 Empty Empty))

findNodes t2 ⟶ [11]
```