დასკვნითი გამოცდა 11.02.2022

კურსის დასახელება: ფუნქციონალური დაპროგრამება **სტატუსი:** ძირითადი
ქულების განაწილება: 1 - **5**ქ., 2 - **5** ქ., 3 - **5** ქ., 4 - **5** ქ., 5 - 1**5** ქ. **ქულათა ჯამი: 40** ქულა
გამოცდის ხანგრძლივობა: 2 საათი.

ბილეთი

1. რეკურსიის გამოყენებით დაწერეთ mymap სახელწოდების მაღალი რიგის საბიბლიოთეკო ფუნქციის განმარტების თქვენი ვერსია, რომელიც ახორციელებს მითითებულ მოქმედებას (ფუნქციას) სიის ყოველ ელემენტზე . განსაზღვრეთ ფუნქციის ტიპი. მოიყვანეთ გამოძახების მაგალითი.

   mymap::(a->b)->[a]->[b]

   mymap fun [] = []

   mymap fun (x:xs) = fun x:mymap fun xs

   mymap (+2) [1,2,5,6,8]

   [3,4,7,8,10]

   it :: [Integer]

   mymap (*5) [21,33,4,6,9]

   [105,165,20,30,45]

   it :: [Integer]

   mymap (>10) [21,4,7,32,8,133]

   [True,False,False,True,False,True]

   it :: [Bool]

2. სიის კონსტრუქტორის გამოყენებით. დაწერეთ მაღალი რიგის საბიბლიოთეკო myfilter ფუნქცია, რომელიც ირჩევს სიიდან ყოველ ელემენტს, რომელიც პრედიკატს აკმაყოფილებს განსაზღვრეთ ფუნქციის ტიპი. მოიყვანეთ გამოძახების მაგალითი.

   myfilter::(a->Bool)->[a]->[a]

```
myfilter pred xs = [x|x<-xs, pred x]


myfilter even [1,2,4,65,7,11,8]

[2,4,8]

it :: [Integer]


myfilter odd [1,21,44,6,77,11,8,50]

[1,21,77,11]

it :: [Integer]


myfilter (>30) [1,21,44,6,77,11,8,50]

[44,77,50]

it :: [Integer]

myfilter (/="test") ["test","answer"]

["answer"]

it :: [[Char]]
```

3. განსაზღვრეთ რეკურსიულად და სიის კონსტრუქტორის გამოყენებით ფუნქცია, რომელიც შესასვლელზე ღებულობს სიას და აბრუნებს ორ მნიშვნელობას: თავდაპირველი სიიდან ლუწი რიცხვების სიას გაერთიანებულს კენტი რიცხვების სიასთან და დადებითი რიცხვების სიას გაერთიანებულს კენტი რიცხვების სიასთან თავდაპირველ სიაში მათი თანმიმდევრობის შენარჩუნებით. განსაზღვრეთ ფუნქციის ტიპი და მოიყვანეთ გამოძახების მაგალითი.

```
/////// რეკურსიის გამოყენებით

funOdds :: Integral a => [a] -> [a]

funOdds [] = []

funOdds (x:xs) |odd x = x:funOdds xs
```

```
        |otherwise = funOdds xs
funEvens :: Integral a => [a] -> [a]
funEvens [] = []
funEvens (x:xs) |even x = x:funEvens xs
        |otherwise = funEvens xs

funPos :: (Ord a, Num a) => [a] -> [a]
funPos [] = []
funPos (x:xs) | x > 0 = x:funPos xs
        |otherwise = funPos xs
funNegative :: (Ord a, Num a) => [a] -> [a]
funNegative [] = []
funNegative (x:xs)  |x > 0 = funNegative xs
        |otherwise = x:funNegative xs

fun :: Integral a => [a] -> ([a], [a])
fun xs = (funEvens xs ++ funOdds xs, funPos xs ++ funNegative xs)
*Main> fun [1,2,3,5,4,8,-4,7,-9]
([2,4,8,-4,1,3,5,7,-9],[1,2,3,5,4,8,7,-4,-9])
it :: ([Integer], [Integer])

///// სიის კონსტრუქტორის გამოყენებით.
fun :: Integral a => [a] -> ([a], [a])
fun xs = ([x|x<-xs, even x] ++ [x|x<-xs, odd x], [x|x<-xs,x > 0] ++ [x|x<-xs,x < 0])
*Main> fun [111,23,35,58,43,8,-42,72,-9,-27,-11]
```

([58,8,-42,72,111,23,35,43,-9,-27,-11],[111,23,35,58,43,8,72,-42,-9,-27,-11])

it :: ([Integer], [Integer])

4. განსაზღვრეთ მაღალი რიგის *all* და *dropWhile* ფუნქციები სტანდარტულ prelude ფაილში მოცემული მათი აღწერების გამოუყენებლად.

```
mymap::(a->b)->[a]->[b]

mymap fun [] = []

mymap fun (x:xs) = fun x:mymap fun xs

myAll::(a->Bool)->[a]->Bool

myAll p xs = foldr (&&) True (mymap p xs)
```

////////////////////////////////

```
myDropWhile::(a->Bool)->[a]->[a]

myDropWhile p xs = [x|x<-xs,not(p x)]
```

5. უძრავი ქონების სააგენტოში იყიდება ბინები- Flat, ოთახები- Room და კერძო სახლები-House. ბინა ხასიათდება სართულით, ფართობით და სახლის სართულების რაოდენობით. ოთახი ხასიათდება ამის გარდა კიდევ ფართობით (დამატებით მთელი ბინის ფართობისა). კერძო სახლი ხასიათდება მხოლოდ ფართობით. ანუ განსაზღვრულია მონაცემთა ტიპი, რომელიც წარმოადგენს უძრავი ქონების ობიექტებზე ინფორმაციას. ანუ მოცემული გვაქვს ტიპი:

```
data NedvObject = Flat Int Int Int | Room Int Int Int Int | House Int deriving (Eq,Show)
-- Flat სართული, ფართობი, სართულიანობა | Room სართული, ფართობი, სართულიანობ,ა ოთახის ფართობი | House ფართობი
```

მონაცემთა ბაზაში ინახება მნიშვნელობების წყვილები, რომელთაგან პირველი წარმოადგენს უძრავ ობიექტს, მეორე–მის ფასს.

განსაზღვრეთ შემდეგი ფუნქციები:

1) `getFlat` მონაცემთა ბაზიდან ირჩევს ბინებს;

```
getFlat :: [(NedvObject,Int)] -> [(NedvObject,Int)]
getFlat [] = []
getFlat ((Flat x,y,z,p):xs) = (Flat x,y,z,p):getFlat xs
getFlat (_:xs) = getFlat xs
```

2) `getRoomByPrice` - ბაზიდან ირჩევს ოთახებს, რომელთა ფასი მოცემულზე მეტია;

```
getRoomByPrice :: [(NedvObject,Int)] -> Int -> [(NedvObject,Int)]
getRoomByPrice [] _ = []
getRoomByPrice ((Room a b c d,y):xs) price =
    if y>price then (Room a b c d,y):getRoomByPrice xs price
    else getRoomByPrice xs price
```

3) `getExceptBounds`, ირჩევს მონაცემთა ბაზიდან ბინებს, რომლებიც არ მდებარეობს პირველ და ბოლო სართულებზე.

```
getExceptBounds :: [(NedvObject,Int)] -> [(NedvObject,Int)]
getExceptBounds [] = []
getExceptBounds ((Flat x y sart,a):xs) =
    if (x/=sart)&&(x/=1) then (Flat x y sart,a):getExceptBounds xs
    else getExceptBounds xs
getExceptBounds (_:xs) = getExceptBounds xs
```

4) `getByType` - მეორე არგუმენტად გადაეცემა სტრიქონი და ბაზიდან შესაბამის ინფორმაციას იღებს. მაგალითად,

```
*Main> getByType [(Flat 3 100 10,1000),(Room 4 120 9 20,1000),(House 200,1000),(Flat 1
100 10,900)] "Room"
[(Room 4 120 9 20,1000)]
it :: [(NedvObject, Int)]
```

```
getByType :: [(NedvObject,Int)] -> String -> [(NedvObject,Int)]

getByType [] _ = []

getByType (x:xs) typeStr = case x of

    (Flat xx y z,a) -> if typeStr=="Flat" then x:getByType xs typeStr

                else getByType xs t

    (Room xx y z zz,a) -> if typeStr=="Room" then x:getByType xs typeStr

                else getByType xs typeStr

    (House y,a) -> if typeStr=="House" then x:getByType xs typeStr

                else getByType xs typeStr
```

დასკვნითი გამოცდა 11.02.2022