**კურსის დასახელება:** ფუნქციონალური დაპროგრამება **სტატუსი:** ძირითადი

**ქულების განაწილება:** 1 - **5**ქ., 2 - **5** ქ., 3 - **5** ქ., 4 - **5** ქ., 5 - 1**5** ქ. =37 **ქულათა ჯამი:** **40** ქულა

**გამოცდის ხანგრძლივობა: 2 საათი.**

**ბილეთი**

1. სიის კონსტრუქტორის 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 :: (t1 -> t) -> [t1] -> [t] +5

myMap f xs = [f x | x <- xs]

**Prelude>** myMap sqrt [4,9,25,225,81]

[2.0,3.0,5.0,15.0,9.0]

it :: [Double]

1. რეკურსიის საშუალებით. დაწერეთ მაღალი რიგის საბიბლიოთეკო myfilter ფუნქცია, რომელიც ირჩევს სიიდან ყოველ ელემენტს, რომელიც პრედიკატს აკმაყოფილებს განსაზღვრეთ ფუნქციის ტიპი. მოიყვანეთ გამოძახების მაგალითი.

myFilter :: (t -> Bool) -> [t] -> [t] +5

myFilter p [] = []

myFilter p (x:xs) = if (p x) then x:myFilter p xs

else myFilter p xs

**Prelude>** myFilter (>=4) [2,3,4,5,6,7,1]

[4,5,6,7]

it :: [Integer]

1. განსაზღვრეთ რეკურსიულად ან სიის კონსტრუქტორის გამოყენებით ფუნქცია, რომელიც შესასვლელზე ღებულობს ნამდვილ რიცხვების სიას და ითვლის ორ მნიშვნელობას: მათ საშუალო არითმეტიკულს და საშუალო გეომეტრიულს. შეეცადეთ, რომ ფუნქციამ მხოლოდ ერთხელ გადახედოს სიას. განსაზღვრეთ ფუნქციის ტიპი. მოიყვანეთ გამოძახების მაგალითი.

calcAvg (x:xs) = doCalcAvg (x:xs) 0 0 1 +5

doCalcAvg (x:xs) sum length product = doCalcAvg xs (sum+x) (length+1) (product\*x)

doCalcAvg [] sum length product = (sum/length , product\*\*(1/length))

**\*Main>** calcAvg [1,2,3,4,5]

(3.0,2.605171084697352)

it :: (Double, Double)

1. განსაზღვრეთ maRali rigis *any* da *takeWhile* funqciebi standartul prelude failSi mocemuli maTi aRwerebis gamouyeneblad.

any :: (a -> Bool) -> [a] -> Bool +5

any p xs = or (map p xs)

any (odd) [2,4,5,9]

True

it :: Bool

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

myany f [] = False

myany f (x:xs) = f x || any f xs

myany (odd) [2,4,5,9]

True

it :: Bool

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

takeWhile p [] = []

takeWhile p (x:xs)

| p x = x : takeWhile p xs

| otherwise = []

takeWhile (<5) [1, 2, 3, 10, 4, 2]

[1,2,3]

it :: [Integer]

mytakeWhile f = foldr (\x acc -> if f x then x : acc else []) []

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

mytakeWhile (<5) [1, 2, 3, 10, 4, 2]

[1,2,3]

it :: [Integer]

1. 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 sarTuli, farTobi, sarTulianoba | Room sarTuli, farTobi, sarTulianob,a oTaxis farTobi | House ფართობი

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

განსაზღვრეთ შემდეგი ფუნქციები:

1. GetRoom - მონაცემთა ბაზიდან არჩევს ოთახებს; =17

GetRoom :: [(NedvObject,Int)] -> [(NedvObject,Int)]

GetRoom [] = []

GetRoom ((Room a b c d,y):xs) = (Room a b c d,y): GetRoom xs

GetRoom (\_:xs) = GetRoom xs

1. getFlatByPrice - ბაზიდან ირჩევს ბინებს, რომელთა ფასი მოცემულის ტოლია;

getFlatByPrice :: [(NedvObject,Int)] -> Int -> [(NedvObject,Int)]

getFlatByPrice [] = []

getFlatByPrice ((Flat a b c,y):xs) price =

if y == price then (Flat a b c,y): getFlatByPrice xs price

else getFlatByPrice xs price

1. getExceptBounds, irCevs binebs mocemul sarTulze;

getExceptBounds :: [(NedvObject,Int)] -> Int -> [(NedvObject,Int)]

getExceptBounds [] \_ = []

getExceptBounds ((Flat x y z,a):xs) floor =

if x == floor then (Flat x y z,a): getExceptBounds xs floor

else getExceptBounds xs floor

getExceptBounds (\_:xs) floor = getExceptBounds xs floor

1. getByType - მეორე არგუმენტად გადაეცემა სტრიქონი, ფასი და ბაზიდან შესაბამის ინფორმაციას იღებს.

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

getByType [] \_ \_ = []

getByType (x:xs) t p = case x of

(Flat xx y z,a) -> if t == "Flat" && a <= p then x:getByType xs t p

else getByType xs t p

(Room xx y z zz,a) -> if t == "Room" && a <= p then x:getByType xs t p

else getByType xs t p

(House y,a) -> if t == "House" && a <= p then x:getByType xs t p

else getByType xs t p

მაგალითად,

**\*Main>** getByType [(Flat 3 100 10,1000),(Room 4 120 9 20,1000),(House 200,1000),(Flat 1 100 10,900)] "Room" 1001

[(Room 4 120 9 20,1000)]

it :: [(NedvObject, Int)]

it :: [(NedvObject, Int)]