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

**ქულების განაწილება:** 1 - **5**ქ., 2 - **5** ქ., 3 - **5** ქ., 4 - **5** ქ., 5 - 1**5** ქ. 32 **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 changer as = [ changer a | a<-as ] +5**

**mymap changer as**

**| as==[] = []**

**| otherwise = (changer (head as)) : mymap changer (tail as)**

**mymap (\x -> x+1) [1,2,3] = [2,3,4]**

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

**Prelude>** let myfilter p xs = [x | x <- xs, p x] **+5**

**Prelude>** myfilter (>3) [1,2,3,4,5,6,7,8,9,10]

[4,5,6,7,8,9,10]

1. 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.=0
2. gansazRvreT maRali rigis *all* da *dropWhile* funqciebi standartul prelude failSi mocemuli maTi aRwerebis gamouyeneblad.

// ganvsazgvrot all funqcia +5

**all p xs = and [ p x | x <- xs]**

**\*Main> let all p xs = and [ p x | x <- xs]**

**all :: (t -> Bool) -> [t] -> Bool**

**\*Main> all even [1,2,3,4]**

**False**

**it :: Bool**

**any p xs = or [ p x | x <- xs]**

**\*Main> let any p xs = or [ p x | x <- xs]**

**any :: (t -> Bool) -> [t] -> Bool**

**\*Main> any even [1,2,3,4]**

**True**

**it :: Bool**

**// ganvsazgvrot dropwhile**

**Prelude> let dropWhile p xs = [x | x <- xs, not(p x)]**

**Prelude> dropWhile (<3) [1,2,3,4,5,6,7]**

**[3,4,5,6,7]**

**რეკურსიით:**

**dropWhile p [] = []**

**dropWhile p (x:xs)**

**| p x = dropWhile p xs**

**| otherwise = x:xs**

**\*Main> dropWhile (<3) [1,2,3,4,5,6,7]**

**[3,4,5,6,7]**

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. getFlat monacemTa bazidan irCevs binebs;
2. getRoomByPrice - ბაზიდან ირჩევს ოთახებს, რომელთა ფასი მოცემულზე მეტია;
3. getExceptBounds, ირჩევს მონაცემთა ბაზიდან ბინებს, რომლებიც არ მდებარეობს პირველ და ბოლო სართულებზე.
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)]

1. **=17**

**getFlat**:: [(NedvObject,Int)] -> [(NedvObject,Int)]

**getFlat** [] = []

**getFlat** ((Flat x,y):xs) = (Flat x,y): **getFlat** xs

**getFlat** (\_:xs) = **getFlat** xs

**2)**

**getRoomByPrice**:: [(NedvObject,Int)] -> Int -> [(NedvObject,Int)]

**getRoomByPrice** [] \_ = []

**getRoomByPrice** ((Room a b c,y):xs) price =

if y == price then (Room a b c,y): **getRoomByPrice** xs price

else **getRoomByPrice** xs price

**3)**

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

getExceptBounds **[] \_ = 0**

getExceptBounds**((Flat n a1 b1): xs) a2 b2 =**

**if(a1 == 1 && b1 == 100) then n**

**else** getExceptBounds **xs a2 b2**

**4)** getByType: [(NedvObject,Int)] -> Int -> [(NedvObject,Int)]

getByType**((NedvObject n a1 b1): xs)**

**if(n == a1) return a1**