**კურსისდასახელება:**ფუნქციონალური დაპროგრამება **სტატუსი:**ძირითადი

**ქულებისგანაწილება:** 1 -**5**ქ., 2 -**5**ქ., 3 -**5**ქ., 4 -**5**ქ.,5 -1**5**ქ.=31 **ქულათაჯამი:40**ქულა

**გამოცდის ხანგრძლივობა: 2საათი.**

**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] +4

mymap m [] = []

mymap m (x:xs) = m x : mymap m xs

mymap m xs = [m x | x <- xs]

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

ფუნქციის ტიპია :

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

Myfilter p[]=[]

Myfilter p (x:xs) | p x =x:myfilter xs

| otherwise = myfilter xs

მაგალითად,ფუნქცია,რომელიც სიიდან იღებს მის დადებით ელემენტებს,განისაზღვრება ასე:

isPositive x = if x> 0 then True else False

getPositive= myfilter isPositive

isPositive x=x >0

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.

რეკურსიულად:

oddNum [] = [] =0

oddNum (x:xs) | x mod 2 == 1 = x:oddNum xs

| otherwise = oddNum xs

evenNum[]=[]

evenNum(x:xs)|x mod 2 == 1 =x:evenNum xs

|otherwise =

evenNum xs

isPositive :: Integer -> Bool

isPositive x = if x > 0 then True

else False

სიის კონსტრუქტორით:

let oddNum n = [x | x <- [1..n], x mod 2 == 1]

let evenNum n = [x | x <- [1..n], x mod 2 == 0]

zipWith :: ?????????

(oddNum->evenNum->c) -> [oddNum]->[evenNum]->[c] zipWith :: (oddNum->isPositive->d) -> [oddNum]->[evenNum]->[d]

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

all :: (a -> Bool) -> [a] -> Bool all p xs = and [p x | x <- xs] +5

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

dropWhile p [] = []

dropWhile p (x:xs) | p x = dropWhile p xs

| otherwise = x:xs

1. უძრავიქონებისსააგენტოშიიყიდებაბინები- Flat, ოთახები- Roomდაკერძოსახლები-House. ბინახასიათდებასართულით, ფართობითდასახლისსართულებისრაოდენობით. ოთახიხასიათდებაამისგარდაკიდევფართობით (დამატებითმთელიბინისფართობისა). კერძოსახლიხასიათდებამხოლოდფართობით. ანუ განსაზღვრულია მონაცემთატიპი, რომელიცწარმოადგენსუძრავიქონებისობიექტებზეინფორმაციას. ანუ მოცემული გვაქვს ტიპი:

data NedvObject = Flat Int Int Int | Room Int Int Int Int | House Int deriving (Eq,Show)

-- Flat sarTuli, farTobi, sarTulianoba | RoomsarTuli, farTobi, sarTulianob,a oTaxis farTobi | House ფართობი

მონაცემთაბაზაშიინახებამნიშვნელობებისწყვილები, რომელთაგანპირველიწარმოადგენსუძრავობიექტს, მეორე–მისფასს.

განსაზღვრეთშემდეგიფუნქციები:

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. getFlats :: [(NedvObject,Int)] -> [(NedvObject,Int)] =17

getFlats [] = []

getFlats ((Flat x,y):xs) = (Flat x,y):getFlats xs

getFlats (\_:xs) = getFlats xs

1. 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 getByPrice xs price

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

getExceptBounds [] = []

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

if (x/=z)&&(x/=1) then (Flat x y z,a):getExceptBounds xs

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

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

getByType [] \_ = []

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

else getByType xs t

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

else getByType xs t

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

else getByType xs t