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

**ქულების განაწილება:** 1 - **5**ქ., 2 - **5** ქ., 3 - **5** ქ., 4 - **5** ქ., 5 - 1**5** ქ. =30 **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 f xs = [f a | a <- xs] +3

myMap f []=[]

myMap f (a:xs) = f a: myMap f 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.

პრედიკატი - ფუნქცია, რომელიც აბრუნებს ბულის მნიშვნელობას +5  
myfilter :: (a -> Bool) -> [a] -> [a]   
myfilter p [] = []   
myfilter p (x:xs) | p x = x : myfilter xs | otherwise = myfilter xs   
   
მაგალითად სიიდან იღებს უარყოფით ელემენტებს  
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.

kentisia :: Integral a => [a] -> [a] =0

kentisia [] = []

kentisia (x:xs)

| odd x = x : kentisia xs

| otherwise = kentisia xs

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

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

all p xs = and (map p xs)

Prelude> all(<11)[1…10]

True

Prelude> all isDigit "123abc"

False

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

dropWhile p [] = []

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

Prelude> dropWhile (<5) [1..10]

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

all a = and \* map a   
any a = or \* map a   
takeWhile \_ [] = [] takeWhile a (x : xs) | a x = x : takeWhile a xs | otherwise = []   
dropWhile \_ [] = [] dropWhile a (x : xs) | a x = dropWhile a xs | otherwise = x : xs

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 d,y):xs) price =

if y>price then (Room a b c d,y): getRoomByPrice xs price

else getRoomByPrice xs price

**3**

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

**4**

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

--

--getHouses [(Flat 3 100 10,1000),(Room 4 120 9 20,500),(House 200,2000),(Flat 1 100 10,900)]

--getByPrice [(Flat 3 100 10,1000),(Room 4 120 9 20,500),(House 200,2000),(Flat 1 100 10,900)] 1001

--getByLevel [(Flat 3 100 10,1000),(Room 4 120 9 20,500),(House 200,2000),(Flat 1 100 10,900)] 1

--getExceptBounds [(Flat 3 150 10,1000),(Room 4 120 9 20,500),(House 200,2000),(Flat 1 100 10,900)]

--query [(Flat 3 110 10,1000),(Room 4 120 9 20,500),(House 200,2000),(Flat 1 100 10,900)] [(MinArea 101),(Type "Flat"),(NeedFloor 3)]