Logičko i funkcijsko programiranje

HASKELL

Kolokvijum, 2024/25

Na **Desktop**-u u direktorijumu **Rad** kreirati direktorijum **ImePrezime_BrIndeksa** i unutar njega sačuvati rešenja datih zadataka.

1. **[4 poena]** Definisati funkciju **pack** koja pakuje listu članova tako što od uzatopnih istih članova formira podlistu.

```
*Main> pack [1,1,1,2,2,3,3,3,1,1,4,4,4,4,4,4]
[[1,1,1],[2,2],[3,3,3],[1,1],[4,4,4,4,4,4]]

*Main> pack ['a','a','a','b','c','c','a','a','d','e','e','e','e']
["aaa","b","cc","aa","d","eeee"]
```

```
pack :: (Eq a) => [a] -> [[a]]
pack [] = []
pack (x:xs) =
       (x : takeWhile (==x) xs) : pack (dropWhile (==x) xs)
```

2. **[4 poena]** Definisati rekurzivnu funkciju **slice**, bez korišćenja ugrađenih funkcija, koja iz liste izdvaja podlistu između dve zadate pozicije.

```
*Main> slice "abcdefgh" 3 6
"cdef"
*Main> slice [1,2,3,4,5] 3 1
[]
*Main> slice [1,2,3,4,5] 1 3
[1,2,3]
*Main> slice "abcdefgh" 3 13
"cdefgh"
```

```
slice :: [a] -> Int -> Int -> [a]
slice [] _ _ = []
slice _ _ 0 = []
slice (x:xs) 1 n = x:slice xs 1 (n-1)
slice (_:xs) m n = slice xs (m-1) (n-1)
```

3. Za potrebe sortiranja liste stringova prema dužini stringa
    a. **[2 poena]** Definisati funkciju **`mapLength`** koja od liste stringova formira listu parova u kojima je prvi element dužina stringa, a drugi element sam string
    ```
    *Main> mapLength ["abc","de","fgh","ij","k"]
    [(3,"abc"),(2,"de"),(3,"fgh"),(2,"ij"),(1,"k")]
    ```

```
mapLength :: [[Char]]  -> [(Int,[Char])]
mapLength = map (\x -> (length x, x))
```

    b. **[3 poena]** Definisati funkciju **`sortPair`** koja sortira listu parova prema vrednosti prvog elementa u paru.
    ```
    *Main> iSort [(3,"abc"),(2,"de"),(3,"fgh"),(2,"ij"),(1,"k")]
    [(1,"k"),(2,"de"),(2,"ij"),(3,"abc"),(3,"fgh")]
    ```

```
iSort :: [(Int,[Char])] -> [(Int,[Char])]
iSort [] = []
iSort ((x,s) :xs) = ins (x,s) (iSort xs)

ins :: (Int,[Char]) -> [(Int,[Char])] -> [(Int,[Char])]
ins (x,s) [] = [(x,s)]
ins (x,s1) ((y,s2):ys)
    |  x <= y = (x,s1) : ((y,s2) : ys)
    |  otherwise = (y,s2) : ins (x,s1) ys
```

    c. **[2 poena]** Definisati funkciju **`lsort`** koja sortira listu stringova prema dužini stringa
    ```
    *Main> lSort ["abc","de","fgh","ij","k"]
    ["k","de","ij","abc","fgh"]
    ```

```
lSort :: [[Char]] -> [[Char]]
lSort = map snd . iSort.mapLength
```

    d. **[bonus 3 poena]** Unaprediti prethodne funkcije tako da rade sa ma kojim tipom u podlistama
    ```
    *Main> lSort [[1,2,3],[4,5],[6,7,8],[9,10],[11]]
    [[11],[4,5],[9,10],[1,2,3],[6,7,8]]
    ```

4. Definisati tip podataka **`Tacka`** za reprezentaciju tačke u koordinatnom sistemu i **`Putanja`** za predstavljanje izlomljene linije određene nizom tačaka. Sve naredne funkcije definisati koristeći ove tipove podataka

```
type Tacka = (Float , Float)
type Putanja = [Tacka]
```

a. **[1 poen]** Definisati funkcije **tacka** i **putanja** koje za 2 broja vraćaju vrednost tipa **Tacka** i **putanja** koja za niz tačaka vreće **Putanju**

```
*Main> tacka 3 5
(3.0,5.0)

*Main> putanja [(2,3),(4,5)]
[(2.0,3.0),(4.0,5.0)]
```

```
tacka :: Float -> Float -> Tacka
tacka x y = (x,y)

putanja :: [Tacka] -> Putanja
putanja = id
```

b. **[3 poena]** Definisati funkcije **translirajTacku** i **translirajPutanju** koje za odgovarajući vektor translira tačku, odnosno putanju

```
*Main> translirajTacku (3,4) 2 3
(5.0,7.0)

*Main> translirajPutanju [(3,4),(5,6),(7,-1)] 2 3
[(5.0,7.0),(7.0,9.0),(9.0,2.0)]
```

```
translirajTacku :: Tacka -> Float -> Float -> Tacka
translirajTacku (x,y) xt yt = tacka (x + xt) (y + yt)

translirajPutanju :: Putanja -> Float -> Float -> Putanja
translirajPutanju putanja x y =
               map (\t -> translirajTacku t x y) putanja
```

c. **[4 poena]** Definisati funkcije **kvadrantTacke** koje određuje kojem kvadrantu pripada tačka i **kvadrantPutanje** koja vraća kvadrant ukoliko se čitava putanja nalazi u tom kvadrantu, a u suprotnom 0

```
*Main> kvadrantTacke (2,3)
1

*Main> kvadrantPutanje [(2,-3),(4,-1),(6,6),(5,-0.5)]
0
```

```
*Main> kvadrantPutanje [(2,-3),(4,-1),(6,-6),(5,-0.5)]
4
kvadrantTacke :: Tacka -> Int
kvadrantTacke (x,y)
        | x > 0 && y > 0 = 1
        | x < 0 && y > 0 = 2
        | x < 0 && y < 0 = 3
        | x > 0 && y < 0 = 4
        | otherwise = 0

kvadrantPutanje :: Putanja -> Int
kvadrantPutanje lst = if istiKvadranti then head kvadranti else 0
             where kvadranti = map kvadrantTacke lst
                   istiKvadranti = all (== head kvadranti) (tail
kvadranti)
```