Задачи за подготовка за второ контролно по ФП,
специалност Компютърни науки

Задача 1. Да се напише на езика Haskell функция **`reverseOrdSuff :: Int -> Int`**, която по дадено естествено число **`k`** намира число, получено от цифрите на най-дългия строго низходящ суфикс на **`k`**, взети в обратен ред.
Примери:
`reverseOrdSuff 37563` → **`36`**
`reverseOrdSuff 32763` → **`367`**
`reverseOrdSuff 32567` → **`7`**
`reverseOrdSuff 32666` → **`6`**

Задача 2. Да се напише на Haskell функция **`sumUnique :: [[Int]] -> Int`**, която по списък от списъци от цели числа намира сумата на тези от числата, които са уникални в рамките на списъка, в който се срещат.
Примери:
`sumUnique [[1,2,3,2],[-4,-4],[5]]` → **`9 (= 1+3+5)`**
`sumUnique [[2,2,2],[3,3,3],[4,4,4]]` → **`0`**
`sumUnique [[1,2,3],[4,5,6],[7,8,9]]` → **`45`**

Задача 3. Продукт се представя с наредена двойка от вида (име, цена). Наличността в даден магазин се представя със списък от продукти.
```
type Product = (String,Double)
type StoreAvailability = [Product]
```
а) Да се напише на Haskell функция
`closestToAverage :: StoreAvailability -> String`, която намира името на продукта, чиято цена е най-близка до средната цена за всички продукти. Ако има повече от един такъв продукт, функцията да връща името на кой да е от намерените.
б) Да се напише на Haskell функция
`cheaperAlternative :: StoreAvailability -> Int`, която намира броя на продуктите, за които има продукт със същото име, но по-ниска цена.
Примери:
```
store1=[("bread",1),("milk",2.5),("lamb",10),("cheese",5),("butter",2.3)]
closestToAverage store1 → "cheese"
store2=[("bread",1),("cheese",2.5),("bread",1),("cheese",5),("butter",2.3)]
cheaperAlternative store2 → 1
```

Задача 4. Нека е даден списък от точки в тримерно пространство, представен като списък от наредени тройки. Да се напише на Haskell функция
`minDistance :: [(Double,Double,Double)] -> Double`, която намира най-малкото от разстоянията между двойките точки от списъка.
Разстоянието d се дефинира по следния начин: ако разглеждаме точките p1=(x1, y1, z1) и p2=(x2, y2, z2), то d(p1, p2) = (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2).

Задача 5. Да се напише на Haskell функция **`reduceStr str`**, която за даден символен низ **`str`** връща негов редуциран вариант, получен в резултат на последователно взаимно унищожаване на двойки съседни знакове, които представляват съответно главен и малък (или малък и главен) вариант на една и съща буква от латиницата. Правилото за взаимно унищожаване на съответни главни и малки (или малки и главни) букви се изпълнява многократно, докато е възможно, върху резултата от последното му прилагане. Всички останали знакове в низа остават непроменени.
Пример:
`reduceStr "dabAcCaCBAcCcaDD"` → **`"dabCBAcaDD"`**

("dabAcCaCBAcCcaDD" → "dabAaCBAcCcaDD" → "dabCBAcCcaDD" → "dabCBAcaDD")

Задача 6. Да се дефинира функция
maximize :: (Ord a, Num a) => [(a -> a)] -> (a -> a), за която оценката на обръщението **maximize l**, където **l** е непразен списък от едноместни числови функции, да е едноместна числова функция на аргумент *x*, която дава стойността *f(x)* на тази функция *f* от списъка **l**, за която числото *f(x)* е най-голямо по абсолютна стойност.
Пример:
Ако **fn = maximize [(\x -> x*x*x),(\x -> x+1)]**,
то **fn 0.5 → 1.5**, а **fn (-2) → -8**

Задача 7. Функцията g е обратна на функцията f в дадено множество A, ако f . g = id в A и g . f = id в A. Да се напише на езика Haskell функция
inverseFun :: (Int -> Int) -> (Int -> Int) -> Int -> Int -> Bool, която за дадени целочислени функции f и g връща **True** точно когато **g** е обратна на **f** в даден целочислен интервал **[a, b]**.
Примери:
inverseFun (\x -> x+1) (\x -> x-1) 5 10 → True
inverseFun (\x -> x*x) (\x -> x^3) 0 1 → True
inverseFun (\x -> x+1) (\x -> x+2) 0 1 → False

Задача 8. Нека е дефиниран алгебричен тип
data BTree = NullT | Node (Float,Float) BTree BTree, който се използва за представяне на двоично дърво от двойки от реални числа, задаващи начала и краища на числови интервали (предполага се, че първият елемент на всяка двойка е по-малък от нейния втори елемент). Напишете на езика Haskell функция **orderedTree tree**, която проверява дали дадено двоично дърво **tree** от тип **BTree** е наредено относно релацията „подинтервал”.
Примери:
1. Ако
```
tree = Node (3.0,10.0) (Node (5.0,8.0) (Node (6.0,7.0) NullT NullT)
                                       (Node (4.0,9.0) NullT NullT))
                       (Node (2.0,12.0) NullT
                                        (Node (1.0,15.0) NullT NullT),
```
то **orderedTree tree → True**.
2. Ако
```
tree = Node (3.0,10.0) (Node (5.0,8.0) (Node (6.0,7.0) NullT NullT)
                                       (Node (7.0,9.0) NullT NullT))
                       (Node (2.0,12.0) NullT
                                        (Node (1.0,15.0) NullT NullT),
```
то **orderedTree tree → False**.