КОНТРОЛНА РАБОТА № 2 ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ
КН, 2-ри курс, 2-ри поток (14.01.2023 г.)

Задача 1

Да се дефинира функция **`prodEvens :: [a] -> a`**, която приема списък от числа и намира произведението на числата, намиращи се на позиции с четен индекс в списъка. Списъкът е индексиран от `0`. Функцията да се дефинира **на функционално ниво** и в решението да се използва **foldr**!

Примери:

```
prodEvens [1,2,3,4,5,6]   → 15
prodEvens [7.66,7,7.99,7] → 61.2034
```

Задача 2

Наети сте от фирма, произвеждаща електрически гаражни врати. Инцидентите с настоящата продуктова линия са довели до множество повредени коли, навехнати крайници и няколко уплашени домашни любимци. Задачата Ви е да напишете по-безопасна версия на софтуера, контролиращ вратите.

Характеристиката на процеса е следната:

- Дистанционното управление има точно един бутон - `P`.
- Винаги се започва със затворена врата.
- Ако вратата е затворена, натискането започва да я отваря, а ако е отворена - да я затваря.
- Отнема 5 секунди, за да се отвори или затвори вратата напълно.
- Докато вратата се движи, едно натискане спира движението, а последващо възобновява движението в същата посока.
- Вратата има сензор, с който открива препятствия. Когато вратата открие препятствие, тя трябва незабавно да обърне посоката на движение.
- Вратата започва да се движи веднага, следователно нейната позиция се променя в същата секунда, в която се случва събитието.

Да се дефинира функция **`controller :: String -> String`**, която приема низ, в който всеки знак представя една секунда, със следните възможни стойности.

- `'.'`: Няма събитие.
- `'P'`: Бутонът `P` е натиснат.
- `'O'`: Открито е препятствие.

Например, `'..P....'` означава, че нищо не се случва в продължение на две секунди, след това бутонът е натиснат и няма други събития.

Функцията трябва да върне низ, в който всеки знак показва позицията на вратата за всяка секунда. Възможните позиции варират от `0` (напълно затворена) до `5` (напълно отворена).

Примери:

```
controller ""                        →  ""
controller "........."               → "0000000000"
controller "P...."                   → "12345"
controller "P.P.."                   → "12222"
controller "..P...O..."              → "0012343210"
controller "P......P......"          → "12345554321000"
controller "P.P.P...."               → "122234555"
controller ".....P.P........P...."   → "00000122222222234555"
controller ".........."              → "0000000000"
controller "P.."                     → "123"
controller "P...."                   → "12345"
controller "P......P......"          → "12345554321000"
controller "P.P.."                   → "12222"
controller "P.P.P...."               → "122234555"
controller ".....P.P........P...."   → "00000122222222234555"
controller ".....P......P.P..P...."  → "000001234555433321000"
controller "P.O...."                 → "1210000"
controller "P......P.O...."          → "12345554345555"
controller "P..OP..P.."              → "1232222100"
controller "P......P..OP..P..."      → "12345554233334555"
controller "..P...O....."            → "001234321000"
```

Задача 3

„Състезател, чийто точки са поне толкова, колкото са точките на завършилия на k-то място състезател, ще премине към следващия кръг, стига точките му да са положително число.“ — извадка от правилата на състезание.

Да се дефинира функция **`numAdvance :: Int -> ([a] -> Int)`**, която приема естествено число `k`, и връща нова анонимна функция, приемаща списък от числа a_1, a_2, ..., a_n ($n \geq k$). Всеки елемент a_i на този списък е резултатът, спечелен от участника, класирал се на `i`-то място. Дадената последователност е ненарастваща (т.е. $\forall\ i \in$ `[0 .. n-1]` е изпълнено $a_i \geq a_{i+1}$). Резултатът от обръщение към новата анонимна функция да е броят участници, които преминават към следващия кръг.

Примери:

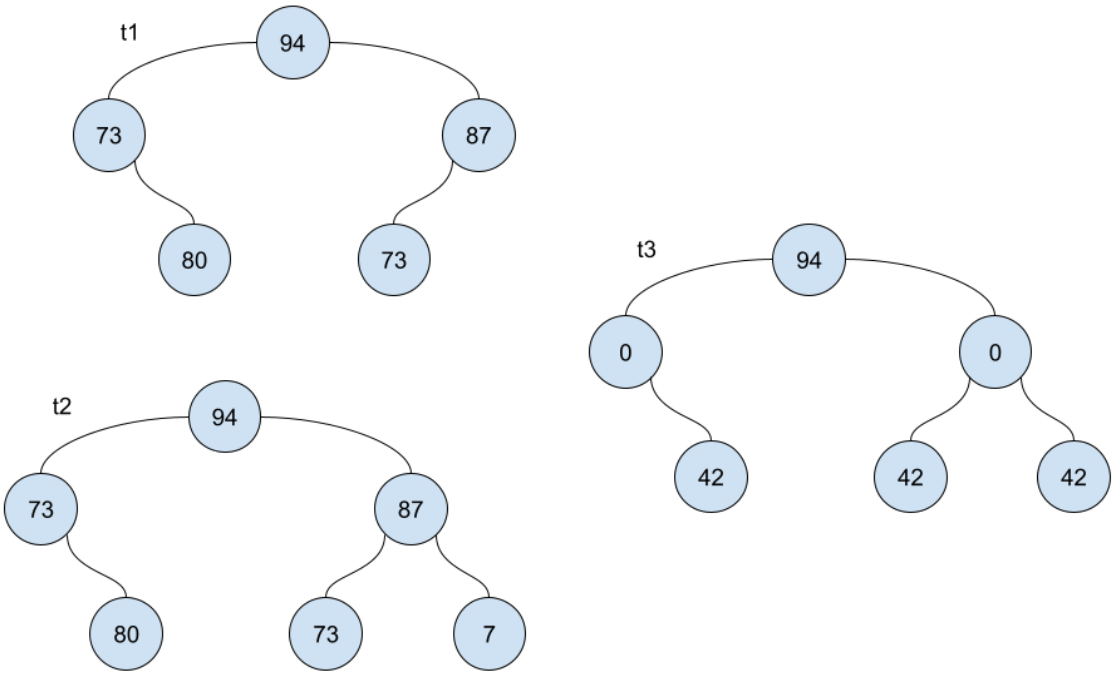
```
(numAdvance 5) [10, 9, 8, 7, 7, 7, 5, 5]       → 6
(numAdvance 2) [0, 0, 0, 0]                     → 0
(numAdvance 3) [10, 9, 8, 7, 7, 7, 5, 5]       → 3
```

```
(numAdvance 1) [10, 9, 8, 7, 7, 7, 5, 5]       → 1
(numAdvance 2) [10, 9, 8, 7, 7, 7, 5, 5]       → 2
(numAdvance 9) [5, 5, 5, 3, 3, 3, 0, 0, 0, 0]  → 6
(numAdvance 10) [5, 5, 5, 3, 3, 3, 0, 0, 0, 0] → 6
```

Задача 4

Дефиниран е полиморфен алгебричен тип **`BTree a`**, описващ двоично дърво:
`data BTree a = Nil | Node a (BTree a) (BTree a)`.
Да се дефинира функция **`maximumLevel :: BTree a -> Int`**, която намира нивото на дървото, на което сумата от стойностите във възлите е максимална. Коренът на дървото се намира на ниво 1. При няколко нива с еднаква сума, която е и максимална, да се връща нивото, което е най-отдалечено от корена.

Примери:




```
t1 = Node 94 (Node 73 Nil (Node 80 Nil Nil)) (Node 87 (Node 73 Nil
Nil) Nil)
t2 = Node 94 (Node 73 Nil (Node 80 Nil Nil)) (Node 87 (Node 73 Nil
Nil) (Node 7 Nil Nil))
t3 = Node 94 (Node 0 Nil (Node 42 Nil Nil)) (Node 0 (Node 42 Nil
Nil) (Node 42 Nil Nil))

maximumLevel t1 → 2
```

```
maximumLevel t2 → 3
maximumLevel t3 → 3
```

Вариант за студента, който няма възможност поради религиозните причини

Задача 1

Да се дефинира предикат **`isPerfectSq :: Int -> Bool`**, който за подадено естествено число проверява дали то е точен квадрат. Да се реализира линеен **итеративен** процес. При невалиден вход да се извежда грешка.

Примери:

```
isPerfectSq 9    → True
isPerfectSq 18   → False
isPerfectSq (-1) → error "Argument has to be a natural number!"
```

Задача 2

Даден е списък от двойки, представящи оценките на ученици от даден клас по даден предмет. Първият елемент на всяка двойка е номерът на ученика в класа, а вторият - получената оценка по предмета (приемаща стойности от `0` до `100`).

Да се дефинира функция **`studAvg :: [(Int, Double)] -> [(Int, Double)]`**, която приема списък от горепосочения вид и изчислява средната стойност на най-добрите пет оценки на всеки ученик. Резултатът да е сортиран спрямо номерата на учениците. Може да се приеме, че всеки ученик има поне пет оценки.

Примери:

```
studAvg [(1, 100), (1, 50), (2, 100), (2, 93), (1, 39), (2, 87),
(1, 89), (1, 87), (1, 90), (2, 100), (2, 76)] → [(1, 83.2), (2,
91.2)]
studAvg [(3, 55), (2, 50), (1, 21), (3, 53), (2, 48), (1, 3), (3,
4), (2, 28), (1, 10), (3, 80), (2, 68), (1, 15), (3, 91), (2, 45),
(1, 49)] → [(1,19.6),(2,47.8),(3,56.6)]
```

Задача 3

Нека разгледаме думата *`abode`*. В нея буквата *`a`* е на позиция *`1`*, а *`b`* е на позиция *`2`*. В английската азбуката *`a`* и *`b`* също са на позиции съответно *`1`* и *`2`*. Забелязваме също, че *`d`* и *`e`* в *`abode`* заемат позициите, които биха заели в азбуката: съответно *`4`* и *`5`*.

Да се дефинира функция **`solve :: [String] -> [Int]`** , която приема списък от думи и връща списък с броя букви, които заемат своите позиции в азбуката за всяка дума. Входните данни ще бъдат съставени само от думи, включващи главни и малки букви от английската азбука.

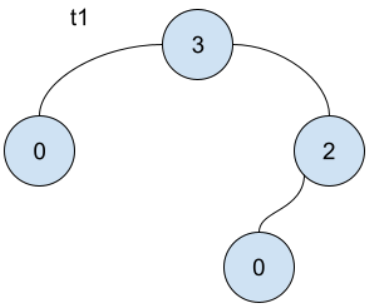
Примери:

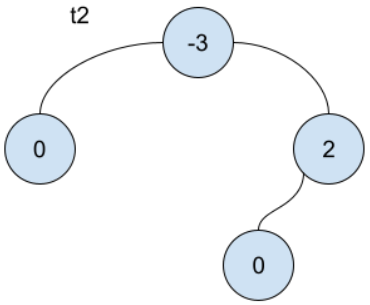
```
solve ["abode","ABc","xyzD"]                    → [4,3,1]
solve ["abide","ABc","xyz"]                     → [4,3,0]
solve ["IAMDEFANDJKL","thedefgh","xyzDEFghijabc"] → [6,5,7]
solve ["encode","abc","xyzD","ABmD"]            → [1, 3, 1, 3]
```

Задача 4

Дефиниран е полиморфен алгебричен тип **`BTree a`**, описващ двоично дърво:
`data BTree a = NullT | Node a (BTree a) (BTree a)`.
Да се дефинира функция **`maxSumSubT :: BTree a -> a`**, която по подадено двоично дърво намира максималната сума на поддърво на това дърво. Сума на дърво е сумата от всички стойности на възли в дървото. Сумата на празното дърво е `0`. Счита се, че всяко дърво е свое поддърво.

Примери:






```
t1 = Node 3 (Node 0 NullT NullT) (Node 2 (Node 0 NullT NullT)
NullT)
t2 = Node (-3) (Node 0 NullT NullT) (Node 2 (Node 0 NullT NullT)
NullT)

maxSumSubT t1 → 5
maxSumSubT t2 → 2
```