ИЗПИТ ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ
Специалност „Информационни системи“, 21.07.2020 г.

Задача 1. Да се дефинира функция `removeEveryKth :: Int -> [a] -> [a]`, която получава естествено число `k` и списък `xs` и премахва всеки `k`-ти елемент на `xs`.

Примери:

```
removeEveryKth 3 [1,2,3,4,5,6,7,8,9] → [1,2,4,5,7,8]
removeEveryKth 4 [1,2,3,4,5,6,7]     → [1,2,3,5,6,7]
```

Задача 2. Според основната теорема на аритметиката, всяко естествено число, по-голямо от 1, може да се представи като произведение на прости числа. Да се дефинира функция `factorize :: Int -> [Int]`, която приема естествено число, по-голямо от 1, и връща сортиран списък от елементите на това представяне.

Примери:

```
factorize 13  → [13]
factorize 152 → [2,2,2,19]
factorize 123 → [3,41]
```

Задача 3. Да се дефинира функция `poly :: [Int] -> (Int -> Int)`, която за даден списък от цели числа [a_0, a_1, a_2, … , a_n] връща функция, чиято стойност за всяко цяло число x е равна на $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$.

Примери:

```
(poly [0,1,2,3]) 1   → 6
(poly [1,1,1,1,1]) 2 → 31
(poly [1,0,1,0]) 3   → 10
(poly [1,2,0,2,1]) 2 → 37
```

Задача 4. Нека представим двоично дърво чрез конструкцията

```
data BTree = Empty | Node Int BTree Btree .
```

Да се дефинира функция `deepestLeavesSum :: BTree -> Int`, която връща сумата на най-дълбоките (най-отдалечените от корена) възли в дадено двоично дърво.

Примери:

```
t3 :: BTree                                          --        1
t3 = Node 1 (Node 2 (Node 4 (Node 7 Empty Empty)--       / \
                            Empty)               --      2   3
                    (Node 5 Empty                --     / \   \
                            Empty))              --    4   5   6
            (Node 3 Empty                        --   /         \
                    (Node 6 Empty                -- 7           8
                            (Node 8 Empty Empty)))
                                                 --        1
t4 :: BTree                                      --       / \
t4 = Node 1 (Node 2 (Node 4 Empty Empty)         --      2   3
                    Empty)                       --     /
            (Node 3 Empty Empty)                 --    4
```

deepestLeavesSum t3 → 15 (7 + 8)

deepestLeavesSum t4 → 4