**Второ контролно по Функционално програмиране**
**Специалност „Информационни системи“, 19.06.2020 г.**

**Задача 1.** Да се дефинира функция **`rotate :: Int -> [a] -> [a]`**, която получава цяло число **`n`** и списък **`xs`** и „завърта“ **`xs`** с **`n`** позиции наляво, т.е. елементите на **`xs`** се преместват с **`n`** позиции наляво, като тези, които при преместването излизат извън списъка, се добавят на края му. При подаване на отрицателно число **`n`**, завъртането е надясно с абсолютната стойност на **`n`**.
*Примери*

```
rotate 3    ['a','b','c','d','e','f','g','h'] → "defghabc"
rotate (-2) ['a','b','c','d','e','f','g','h'] → "ghabcdef"
```

**Задача 2.** Нека имаме следното представяне на двоично дърво от цели числа:

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

Казваме, че едно двоично дърво е *огледално-симетрично*, ако лявото му поддърво е огледален образ на дясното. Да се дефинира предикат **`isSymmetric :: BTree -> Bool`**, който проверява дали дадено двоично дърво е *огледално-симетрично*.
*Примери*

```
t3 :: BTree                          --   1
t3 = Node 1 (Node 2 Empty Empty)     --  / \
            (Node 3 Empty Empty)     -- 2   3

t4 :: BTree                                  --     1
t4 = Node 1 (Node 2 (Node 3 Empty Empty)     --    / \
                    Empty)                   --   2   2
            (Node 2 Empty                    --  /     \
                    (Node 3 Empty Empty))    -- 3       3

t5 :: BTree                                            --       1
t5 = Node 1 (Node 2 (Node 3 Empty Empty)               --    /     \
                    (Node 7 (Node 5 Empty Empty)       --   2       2
                            Empty))                    --  / \     / \
            (Node 2 (Node 7 Empty                      -- 3   7   7   3
                            (Node 5 Empty Empty))      --    /     \
                    (Node 3 Empty Empty))              --   5       5
```

```
isSymmetric t3 → False
isSymmetric t4 → True
isSymmetric t5 → True
```

**Задача 3.** Стандартните списъци в *Haskell* са хомогенни, т.е. съдържат елементи от един и същ тип. Нека дефинираме наш тип „вложен списък“ **`NestedList`**, който може да съдържа както „обикновени“ („атомарни“) елементи, така и други списъци от типа **`NestedList`**. За леснота елементите ще са само цели числа.

```
data NestedList = Elem Int | List [NestedList]
```

Да се дефинира функция **`flatten :: NestedList -> [Int]`**, която получава вложен списък **`list`** и го „изглажда“, превръщайки го в стандартен, хомогенен списък, съдържащ числата от **`list`**.

*Примери*

```
flatten
(List [Elem 1, List [Elem 2, List [Elem 3, Elem 4], Elem 5]])
→ [1,2,3,4,5]
flatten (Elem 1) → [1]
```