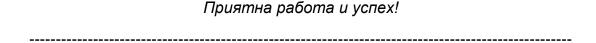
# Домашна работа №3 по Функционално програмиране Специалност "Информационни системи", 2019/2020 учебна година

**Крайният срок** за предаване на домашните работи е **03.06.2020 г.**, 23:55 ч.

Решенията трябва да са готови за компилиране и автоматично тестване. Важно е писмените работи да бъдат добре форматирани и да съдържат коментари на ключовите места.

Предайте решенията на всички задачи в един файл с наименование hw3\_<FN>.hs, където <FN> е Вашият факултетен номер.



Нека е дадено следното представяне на двоично дърво:

```
data BTree a = Empty | Node a (BTree a) (BTree a)
```

**Задача 1.** Да се дефинира функция containsWord :: BTree Char -> String -> Bool, която по дадено двоично дърво от символи и дума, съставена от поне една буква, проверява дали думата се среща в дървото, като последният символ от думата е лист в това дърво.

#### Пример:

Задача 2. Да се дефинира функция genWords :: BTree Char -> [String], която по дадено двоично дърво от символи връща списък от всички думи, съдържащи се в него.

#### Пример:

```
genWords t1 \rightarrow ["acf", "acd", "abe", "cf", "cd", "f", "d", "be", "e"]
```

Задача 3. Да се дефинира функция allContain :: [BTree Char] -> [String], която по даден списък от двоични дървета от символи връща списък от тези думи, които се съдържат във всички дървета.

#### Пример:

### **Задача 4.** Нека е дадено следното представяне на n-арно дърво:

```
data NTree = Nil | NNode Int [NTree]
```

Да се дефинира функция isGraceful :: NTree -> Bool, която приема n-арно дърво и проверява дали то е *грациозно*. Казваме, че едно дърво е *грациозно*, ако абсолютните стойности на разликите между стойностите на всеки елемент и бащиния му са четни.

## Пример:

**БОНУС Задача 5 (първите 4 задачи са достатъчни за оценка 6,00).** Да се дефинира функция isBinarySearchTree :: BTree Int -> Bool, която проверява дали дадено двоично дърво от цели числа е двоично дърво за търсене. Казваме, че едно двоично дърво е двоично дърво за търсене, ако лявото му поддърво съдържа само възли със стойности, по-малки от тази в корена, а дясното

му поддърво - само стойности, по-големи или равни на тази в корена. Освен това, трябва и самите поддървета също да са *двоични дървета за търсене*.

#### Примери:

```
t5 :: BTree Int
          (Node 10 (Node 9 Empty Empty) -- / \
                 (Node 14 Empty Empty)) -- 1 4 9 14
isBinarySearchTree t5 \rightarrow True
t6 :: BTree Int
(Node 10 (Node 5 Empty Empty) -- / \
                  (Node 14 Empty Empty)) -- 1 4 5 14
isBinarySearchTree t6 \rightarrow False (в дясното поддърво има стойности,
                          по-малки от корена)
t7 :: BTree
            Int
t7 = Node 8 (Node 3 (Node 5 Empty Empty) -- /
(Node 6 Empty Empty)) -- 3
                 (Node 6 Empty Empty))
          (Node 10 (Node 5 Empty Empty) -- / \
                 (Node 14 Empty Empty)) -- 5 6 9 14
```

isBinarySearchTree t7  $\rightarrow$  False (лявото поддърво не е BST)