КОНТРОЛНА РАБОТА № 2 ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ
КН, 2-ри курс, 1-ви поток (16.01.2021 г.)

Задача 1. Чрез използването на **n** кубове е построена сграда. Кубът, намиращ се най-отдолу, т.е. основата, е с обем n^3. Кубът, който е върху него, е с обем $(n-1)^3$. Кубът, поставен най-отгоре, има обем 1^3.
Обемът на цялата сграда е: $n^3 + (n-1)^3 + ... + 1^3 = m$.
Да се дефинира функция **findNb :: Integer -> Integer**, която по дадено **m** да връща броя кубове **n**, необходими за построяване на сградата. Ако **n** не съществува, да се връща **-1**.

Примери:

```
findNb 1071225            --> 45
findNb 40539911473216     --> 3568
findNb 135440716410000    --> 4824
findNb 4183059834009      --> 2022
findNb 91716553919377     --> -1
findNb 24723578342962     --> -1
```

Задача 2. Нека са дадени две едноместни целочислени функции **f** и **g** и списък от цели числа **xs**. Ще казваме, че функцията **f** *доминира* **g** върху множеството **xs**, ако за всяко $x \in xs$ е вярно, че $|f(x)| \geq |g(x)|$.
Да се дефинира функция **dominates :: (Int -> Int) -> (Int -> Int) -> [Int] -> Bool**, която връща резултата от проверката дали функцията **f** доминира **g** върху множеството **xs**.

Примери:

```
dominates (+4) (*2) [1..4] --> True
dominates (+4) (*2) [1..5] --> False (5+4=9 < 5*2=10)
```

Задача 3. Нека е дефиниран типът **type Point = (Double,Double)**, представящ точка с реални координати. Да се дефинира функция **splitPoints :: Point -> Double -> [Point] -> ([Point],[Point])**, която приема точка **p**, радиус **r** и списък от точки **ps** и връща като резултат двойка от списъци – такава, че първият съдържа тези точки от **ps**, които са в кръга с център **p** и радиус **r**, а вторият – всички останали точки от **ps**.

Пример:

```
splitPoints (1,1) 5 [(1,2),(2,3),(10,15),(-1,1),(12,14)]
--> ([(1.0,2.0),(2.0,3.0),(-1.0,1.0)],[(10.0,15.0),(12.0,14.0)])
```

Задача 4. Нека е дефинирано следното представяне на двоично дърво:
`data BTree = Empty | Node Int BTree BTree.`
Да се дефинира функция **`isBinarySearchTree :: BTree -> Bool`**, която проверява дали подадено 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:

```
t1 :: BTree                                    --       8
t1 = Node 8 (Node 3 (Node 1 Empty Empty)       --     /   \
                    (Node 4 Empty Empty))      --    3     10
            (Node 10 (Node 9 Empty Empty)      --   / \    / \
                     (Node 14 Empty Empty)) -- 1   4  9   14

t2 :: BTree                                    --       8
t2 = Node 8 (Node 3 (Node 1 Empty Empty)       --     /   \
                    (Node 4 Empty Empty))      --    3     10
            (Node 10 (Node 5 Empty Empty)      --   / \    / \
                     (Node 14 Empty Empty)) -- 1   4  5   14

t3 :: BTree                                    --       8
t3 = Node 8 (Node 3 (Node 5 Empty Empty)       --     /   \
                    (Node 6 Empty Empty))      --    3     10
            (Node 10 (Node 9 Empty Empty)      --   / \    / \
                     (Node 14 Empty Empty)) -- 5   6  9   14
```

`isBinarySearchTree t1 --> True`
`isBinarySearchTree t2 --> False` (в дясното поддърво има стойности, по-малки от тази в корена)
`isBinarySearchTree t3 --> False` (лявото поддърво не е *двоично дърво за търсене*)