# **Упражнения:** *Рекурсия*

## **Зад. 1** *Логаритъм втори от n*

Дефинирайте функция, която приема един параметър - число и връща като резултат логаритъм втори от подаденото число (закръглено до целочислен тип)

**Пример**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 15 | 3 |
| 10 | 3 |
| 10000 | 13 |

### **Подсказки**

1. Дефинирайте функцията като за 1 нека връща резултат 0
2. При n > 1 върнете резултат 1 + резултата рекурсивното извикване на същата функция за n / 2
   1. По този начин се постига цикличен ефект (подобно на for-loop), като началото е n, условието е n > 1, а на всяка итерация n намалява двойно

## **Зад. 2** *Факториел (опашкова рекурсия)*

Дефинирайте функция, която връща като резултат n-факториел. Използвайте алгоритъм с опашкова рекурсия

### **Пример**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 | 120 |
| 10 | 3628800 |

### **Подсказки**

1. Дефинирайте функция findFactorial, която приема 3 параметъра
   1. n - желания факториел
   2. initialValue - началната стойност (факториела започва от 1)
   3. Index - индекс точно, какъвто би се използвал в нормален for-цикъл
2. Проверете дали индекса надвишава желаното число n
   1. Ако да - върнете initialValue
   2. Ако не - рекурсивно извикайте findFactorial като подадете същия n, промените initialValue на стойност равна на сегашната умножено по индекса, увеличете индекса с единица
3. Дефинирайте втора функция factorial, която да служи за помощна и да приема само 1 параметър - n - желания факториел
4. Нека factorial извиква функцията findFactorial като задава за стойности на нейните параметри n за търсеното число, 1 за стартова стойност и 1 като начален индекс

## **Зад. 3** *Фибоначи (опашкова рекурсия)*

Дефинирайте функция, която връща като резултат n-тото число от редицата на Фибоначи. Използвайте алгоритъм с опашкова рекурсия.

### **Пример**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 10 | 55 |
| 21 | 10946 |

### **Подсказки**

1. Дефинирайте функцията findFibonacci. Нека тя приема 4 параметъра - n - желаното по ред число от редицата на Фибоначи, initialValue - стойността, от която редицата започва, prevValue - стойността на предишното по ред число от редицата, index - индексатор, който следи до кое число от редицата сме стигнали
2. Проверете дали индексаторът не е надвишил или е равен на желаното по ред число - n
   1. Ако да - нека функцията върне началната стойност - initalValue
   2. Ако не - нека функцията се извика рекурсивно като стойността на n се запазва, началната стойност вече е равна началната стойност плюс предишното по ред число. За стойност на предишното число вече ни е нужно initialValue, а индекса трябва да се увеличи с единица, за да напредне рекурсията към дъното си
3. Дефинирайте помощна функция fibonacci, която приема един параметър - n и извиква функцията findFibonacci като задава стойности за n - n, за начална стойност - единица, за предишно по ред число - 0 и за индекс - 1

## 

## 

## 

## 

## **Зад. 4** *Обърнат триъгълник*

Дефинирайте функция, която връща приема като параметър число - n и принтира на конзолата обърнат триъгълник от ‘\*’, като на се започне на първи ред с n звездички и на всеки следващ ред принтира с една по-малко. При вход 0 да не се принтира нищо на конзолата

**Пример**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 | \*\*\*\*\*  \*\*\*\*  \*\*\*  \*\*  \* |
| 1 | \* |
| 4 | \*\*\*\*  \*\*\*  \*\*  \* |

### **Подсказки**

1. Дефинирайте помощна функция asterixStringRow, която приема един параметър - броя на символи, които трябва да се повтарят и връща като резултат символен низ с дължина n, съставен от звездички (‘\*’)
   1. Разгледайте как работи вградения метод replace
2. Дефинирайте функцията printTriangle, която приема n
   1. За `printTriangle 0` функцията не трябва да връща нищо (Разгледайте void типа в Haskell - () и го ипозлвайте като резултат от функцията)
   2. За всяко друго n нека функцията принтира резултата от asterixStringRow за n, след което рекурсивно извиква себе си за n - 1