Национална програма
"Обучение за ИТ умения и кариера"
https://it-kariera.mon.bg

Министерството на
образованието и науката
https://www.mon.bg

# Списъци

Функционално програмиране

# Списъци

- Инициализация на списък:

```
x = [1,2,3]
```

- Празен списък:

```
empty = []
```

- Операторът `:`

```
y = 0 : x -- [0,1,2,3]
```

# Списъци

- Инициализация на списък:

```
x = [1,2,3]
```

Елементите в списъка се разделят от запетая и се обграждат от квадратни скоби `[` `]`

- Празен списък:

```
empty = []
```

- Операторът `:`

```
y = 0 : x -- [0,1,2,3]
```

# Списъци

- Инициализация на списък:

```
x = [1,2,3]
```

Операторът : винаги приема елемент от лявата си страна и списък от дясната си страна, като долепя елемента в началото на списъка, но без да променя вече съществуващия списък

- Празен списък:

```
empty = []
```

- Операторът `:`

```
y = 0 : x -- [0,1,2,3]
```

# Списъци

```
x' = 1 : (2 : (3: [])) -- [1,2,3]
```

- Символните низове също са списъци:

```
str = "abcde"
```

```
str' = 'a' : 'b' : 'c' : 'd' : 'e' : []
```

# Списъци

```
x' = 1 : (2 : (3: [])) -- [1,2,3]
```

- Символните низове също са списъци:

```
str = "abcde"
```

```
str' = 'a' : 'b' : 'c' : 'd' : 'e' : []
```

При извикване str и str' имат една и съща стойност - abcde

# Глава и опашка на списъци

- Глава на списъка е първият елемент от него

```
head[1, 2, 3] -- 1
```

- Опашка на списъка е всичко останало освен главата

```
tail [1,2,3] -- [2, 3]
```

- В комбинация от функциите можем да достъпим следващия елемент от списъка

```
head (tail [1,2,3]) -- 2
```

# Рекурсивно обхождане на списък

- Рекурсивно обхождане на списък и умножаване на всяка

```
doubleList list =
    if null list
    then []
    else (2 * (head list) : (doubleList (tail
list)))
```

```
doubleList [1,2,3,4,5] -- [2,4,6,8,10]
```

# Рекурсивно обхождане на списък

- Рекурсивно обхождане на списък и умножаване на всяка

```
doubleList list =
    if null list
    then []
    else (2 * (head list) : (doubleList (tail
list)))
```

Рекурсивното обхождане започва като приема целият списък за параметър

```
doubleList [1,2,3,4,5] -- [2,4,6,8,10]
```

# Рекурсивно обхождане на списък

- Рекурсивно обхождане на списък и умножаване на всяка

```
doubleList list =
    if null list
    then []
    else (2 * (head list) : (doubleList (tail
list)))
```

Дъно на рекурсията е достигането на празен списък - използва се вградената в Haskell функция `null`, която връща съответно True при празен списък и False при непразен

```
doubleList [1,2,3,4,5] -- [2,4,6,8,10]
```

# Рекурсивно обхождане на списък

- Рекурсивно обхождане на списък и умножаване на всяка

```
doubleList list =
    if null list
    then []
    else (2 * (head list) : (doubleList (tail
list)))
```

В случай, че списък не е празен функцията връща като резултат първият елемент от списъка умножен по 2 и рекурсивно се извиква за останалите елементи

```
doubleList [1,2,3,4,5] -- [2,4,6,8,10]
```

# Рекурсивно обхождане на списък

- Функция, филтрираща елементите в списък (премахва

```
removeOdd nums =
    if null nums
    then []
    else
        if (mod (head nums) 2 ) == 0
        then (head nums) : (removeOdd (tail nums))
        else removeOdd (tail nums)
```

Ако даден елемент не отговаря на условието, просто се пропуска и рекурсията се извиква за опашката на списъка

# Дължина на списък

- За намиране на дължината на списък се използва функцията `length`

```
length [1,2,3,4,5] -- 5
```

- Собствена функция за намиране на дължината

```
listLength [] = 0
listLength list = findLength 1 list
findLength length list =
    if null list
    then (length - 1)
    else findLength (length + 1) (tail list)
```

# Създаване на списък чрез рекурсия

- Създаване на списък чрез рекурсия:

```
createList start end = createListLoop [] start end
createListLoop list start end =
    if start > end
    then list
    else createListLoop (list ++ [start]) (start + 1)
end
```

```
createList 1 10 -- [1,2,3,4,5,6,7,8,9,10]
```

# Създаване на списък чрез рекурсия

- Създаване на списък чрез рекурсия:

```
createList start end = createListLoop [] start end
createListLoop list start end =
    if start > end
    then list
    else createListLoop (list ++ [start]) (start + 1)
end
```

`++` операторът добавя елемент в края на списъка

```
createList 1 10 -- [1,2,3,4,5,6,7,8,9,10]
```

# Създаване на списък чрез рекурсия

- Създаване на обърнат списък чрез рекурсия:

```
createReverseList start end = createReverseListLoop []
start end
createReverseListLoop list start end =
    if start > end
    then list
    else createReverseListLoop (start : list) (start + 1)
end
```

```
createReverseList 1 10 -- [10,9,8,7,6,5,4,3,2,1]
```

# Създаване на списък чрез рекурсия

- Създаване на обърнат списък чрез рекурсия:

```
createReverseList start end = createReverseListLoop []
start end
createReverseListLoop list start end
    if start > end
    then list
    else createReverseListLoop (start : list) (start + 1)
end
```

Операторът `:` добавя елемент в началото на списъка

```
createReverseList 1 10 -- [10,9,8,7,6,5,4,3,2,1]
```

# Създаване на списък чрез рекурсия

- В Haskell създаването на списък може да продължава до безкрайност:

```
intsFrom n = n : (intsFrom (n+1))
ints = intsFrom 1
```

```
ints -- Продължава до безкрайност
```

```
take 10 ints -- [1,2,3,4,5,6,7,8,9,10]
```

# Създаване на списък чрез рекурсия

- В Haskell създаването на списък може да продължава до безкрайност:

```
intsFrom n = n : (intsFrom (n+1))
ints = intsFrom 1
```

```
ints -- Продължава
```

Haskell обаче е “мързелив” език - функцията `intsFrom` няма да се изпълнява до безкрайност, а само до там, докъдето създаденият списък е нужен

```
take 10 ints -- [1,2,3,4,5,6,7,8,9,10]
```

# Създаване на списък чрез рекурсия

- В Haskell създаването на списък може да продължава до безкрайност:

```
intsFrom n = n : (intsFrom (n+1))
ints = intsFrom 1
```

```
ints -- Продължава
```

Функцията ще продължи да се извиква рекурсивно чак когато последващите елементи от списъка се използват

```
take 10 ints -- [1,2,3,4,5,6,7,8,9,10]
```

# Задача:

- Дефинирайте функция, която приема списък и число n и връща като резултат n-тия елемент от списъка
- Бележка: Не използвайте вградения в Haskell оператор `!!`

# Решение:

```
nThElement list n = nThElementLoop list (length list) n 0
nThElement [] _ = error "Empty list"
nThElementLoop list listLength n index =
    if n >= listLength || n < 0
    then error "Index outside bounds of array"
    else if index == n
         then (head list)
         else nThElementLoop (tail list) listLength n (index + 1)
```

Национална програма
"Обучение за ИТ умения и кариера"
https://it-kariera.mon.bg

Министерството на
образованието и науката
https://www.mon.bg