Berik Alisher IT-2308

Functional Programming: Assignment 1

Task 1 - Square List

```haskell
squareList :: [Int] -> [Int]
-- "map squareNum" means apply
-- function 'squareNum' each element of list
squareList lst = map squareNum lst

squareNum :: Int -> Int
squareNum x = x * x
```

```haskell
--Square list finction
putStrLn "\n\tSquare List "
putStrLn "Enter list (e.g. [1,2,3]): "
listStr <- getLine
let testList = readListToInt listStr

putStr "Square list: "
print $ squareList testList
```

Results

```
        Square List
Enter list (e.g. [1,2,3]):
[1,2,3,4,4,3]
Square list: [1,4,9,16,16,9]
```

Additional

```haskell
--Just created function bcs
--I got used to write on C++, Java like
readListToInt :: String -> [Int]
readListToInt lst = read lst
```

Task 2 - Fibonacci (Recursive)

```haskell
fibonacci :: Int -> Int
--Base cases
fibonacci 1 = 1
fibonacci 2 = 1
--Recursive case (number greater than 2)
fibonacci n | n > 2 = fibonacci(n-1) + fibonacci(n-2)
```

```haskell
--Fibonacci function
putStrLn "\n\tFibonaci "
putStr "Enter n-th value of Fibonacci (start from 1): "
fibStr <- getLine
let fib = read fibStr :: Int

putStr (fibStr ++ "th value of Fibonacci: ")
print $ fibonacci fib
```

Results

```
        Fibonaci
Enter n-th value of Fibonacci (start from 1): 10
10th value of Fi: 55
```

Task 3 - is Even

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0
```

```haskell
--Even function
putStrLn "\n\tEven "
putStr "Enter x to check Even: "
xStr <- getLine
let x = read xStr :: Int

putStr "Is Even: "
print $ isEven x
```

Results

```
        Even
Enter x to check Even: 23
Is Even: False
```

```
        Even
Enter x to check Even: 24
Is Even: True
```

Task 4 - Sum of List

```haskell
sumList:: [Int] -> Int
-- sum - built-in function
sumList lst = sum lst
```

```haskell
--Sum List function
putStrLn "\n\tSum of List's elements "
putStr "Enter list: "
listStr <- getLine
let testList = readListToInt listStr

putStr "Sum of list: "
print $ sumList testList
```

Results

```
        Sum of List's elements
Enter list: [1,2,3,5]
Sum of list: 11
```

Task 5 - Swap

```haskell
swap :: (a, b) -> (b, a)
swap (x, y) = (y, x)
```

Results

```
        Swap
Enter a: 1253
Enter b: 9524
Swaped:
A: 9524

B: 1253
```

```haskell
--Swap function
putStrLn "\n\tSwap "
putStr "Enter a: "
aStr <- getLine
let a = read aStr :: Int
putStr "Enter b: "
bStr <- getLine
let b = read bStr :: Int


putStrLn "Swaped: "
-- let (a, b) = swap (a, b)
-- This statement output
-- infinite loop exception
-- I still dunno how it happens
let (newA, newB) = swap (a, b)
putStr "A: "
print newA
putStr "\nB: "
print newB
```