

**Functional Programming**

LAB #1

Topic: Laboratory work Nr 1.

Student’s name: Andrey

Student’s surname: Perunov

St. code:

81049

Lecturer: Aleksejs Vesjolijs

**RIGA**

Contents

[Git repository 3](#_Toc163209855)

[Tasks 3](#_Toc163209856)

[Task 1 3](#_Toc163209857)

[Task 2 4](#_Toc163209858)

[Task 3 5](#_Toc163209859)

[Task 4 6](#_Toc163209860)

# Git repository

<https://github.com/AndreyPerunov/FP-Lab1>

# Tasks

## Task 1

**Objective: Write a Haskell program where you define and implement your own version of the map function, named myMap. The map function applies a given function to each element in a list, returning a new list of the results. Your myMap function should replicate this behavior.**

**Requirements:**

* The myMap function must take two inputs: a function and a list.
* The function should be applicable to each element of the list.
* The output should be a new list with the function applied to each element of the original list.
* Do not use the built-in map function in your implementation.

**Variant 1.**

Define and use myMap to increment every number in a list [0, 5, 7, 13, 25, 36, 100, 999].

**Solution:**

list :: [Int]

list = [0, 5, 7, 13, 25, 36, 100, 999]

increment :: Int -> Int

increment x = x + 1

*-- My Map Function*

*-- Input:*

*-- List, [Int]*

*-- Function, (Int -> Int)*

*-- Output:*

*-- List, [Int]*

myMap :: [Int] -> (Int -> Int) -> [Int]

*-- Recursion approach:*

*-- return empty list if list is empty*

myMap [] f = []

*-- apply function to the first element*

*-- apply map function to the rest of the list*

myMap (x:xs) f = (f x : myMap xs f)

*-- List comprehension approach:*

*-- myMap list f = [f head | head <- list]*

main :: IO ()

main = do

  putStrLn "List: "

  print list

  putStrLn "Incremented List: "

  print (myMap list increment)

**Output:**

A screen shot of a computer program

Description automatically generated

## Task 2

**Create program in HASKELL.**

**Variant 1.**

Define function that takes a list of person names as user input.

**Requirements:**

* Return list with all names written in backwards manner. E.g. “Richard” -> “Drahcir”
* All returned names must be starting from the uppercase

**Solution:**

import Data.Char(toUpper)

import Data.Char(toLower)

*-- Reverses the string*

*-- Input: String*

*-- Output: String*

reverseName :: String -> String

*-- if the string is empty, return empty string*

reverseName [] = []

*-- putting the first letter in the end*

*-- calling the function recursively as a first part of the list*

reverseName (x:xs) = reverseName xs ++ [x]

*-- Make the first letter uppercase and the rest lowercase*

*-- Input:*

*-- String*

*-- Output:*

*-- String*

capitalize :: String -> String

capitalize [] = []

capitalize (x:xs) = toUpper x : lowercase xs

  where

    lowercase [] = []

    lowercase (x:xs) = toLower x : lowercase xs

*-- Applies reverseName and capitalize to all elements of the list*

*-- Input:*

*-- List :: [String]*

*-- Output:*

*-- List :: [String]*

reverseNames :: [String] -> [String]

reverseNames [] = []

reverseNames (x:xs) = (capitalize (reverseName x)) : reverseNames xs

main = do

  putStrLn "Enter the list of names: "

  input <- getLine

  let list = words input

  putStrLn "List: "

  print list

  putStrLn "Reversed List: "

  print (reverseNames list)

**Output:**

A screen shot of a computer program

Description automatically generated

## Task 3

**Create program in HASKELL.**

**Requirements:**

Define a function that takes a sentence as user input and returns sentence containing only vowels. Function should be able to accept string with input and print output.

**Variant 1.**

* Enhance function with functionality that allows to reverse order of words in a sentence, including punctuation.

**Solution:**

import Data.Char(toLower)

*-- Returns True if the character is a vowel*

*-- Input:*

*-- Char*

*-- Output:*

*-- Bool*

isVowel :: Char -> Bool

isVowel c = (toLower c) `elem` "aiueo"

*-- Returns True if the character is a punctuation*

*-- Input:*

*-- Char*

*-- Output:*

*-- Bool*

isPunctuation :: Char -> Bool

isPunctuation c = c `elem` "(),.!?{}[]"

*-- Returns sentence containing only vowels*

*-- Input:*

*-- String*

*-- Output:*

*-- String*

onlyVowels :: String -> String

onlyVowels [] = []

onlyVowels (x:xs)

  | isVowel x = x : onlyVowels xs

  | isPunctuation x = x : onlyVowels xs

  | otherwise = onlyVowels xs

*-- Reverses the string*

*-- Input:*

*-- String*

*-- Output:*

*-- String*

reverseString :: String -> String

*-- if the string is empty, return empty string*

reverseString [] = []

*-- putting the first letter in the end*

*-- calling the function recursively as a first part of the list*

reverseString (x:xs) = reverseString xs ++ [x]

main = do

  putStrLn "Enter a sentence: "

  sentence <- getLine

  putStrLn "Only Vowels Sentence, Reversed, With Punctuation: "

  print (reverse (onlyVowels sentence))

**Output:**

A blue and green text on a black background

Description automatically generated

## Task 4

**Create program in HASKELL. Program should take user input and print output.**

**Variant 1-2.**

Define a function that takes two tuples of string type as user input. It should create new tuple, based on user input.

Tuple mapping:

Tuple1 -> (a, b)

Tuple2 -> (c, d)

Output: Tuple3 -> (a, d)

**Solution:**

*-- Creates new tuple based on two tuples*

*-- Input:*

*--   tuple1: ( generic types a, b )*

*--   tuple2: ( generic types c, d )*

*-- Output:*

*--   tuple3: ( generic types a, d)*

newTuple :: (a, b) -> (c, d) -> (a, d)

*-- put first element of the first tuple and second element of the second tuple into the new tuple*

newTuple tuple1 tuple2 = (fst tuple1, snd tuple2)

*-- Reads two strings from the console*

*-- Output:*

*--   tuple: (String, String)*

readTuple :: IO (String, String)

readTuple = do

  a <- getLine

  b <- getLine

  return (a, b)

main :: IO ()

main = do

  putStrLn "Enter first tuple:"

  tuple1 <- readTuple

  putStrLn "Enter second tuple:"

  tuple2 <- readTuple

  let tuple3 = newTuple tuple1 tuple2

  putStrLn $ "Tuple 1: " ++ show tuple1

  putStrLn $ "Tuple 2: " ++ show tuple2

  putStrLn $ "Result tuple: " ++ show tuple3

**Output:**

A screen shot of a computer

Description automatically generated