Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота № 2

з дисципліни «Функційне програмування мовою Haskell»

Виконав:
Студент ФІОТ
Групи ІП-93
Владіміров А.О.

Київ – 2023

Мета: набути досвіду визначення та використання функцій вищого порядку.

Завдання: визначте функції вищого порядку для виконання вказаних завдань. За необхідності завдання можна розширити. Наприклад у завданні 1.1 результат повертати у формі кортежів ('a',3) або (3,'a'), або ('a',3,"a3") тощо. Визначте вказані функції в кожному з завдань: а) без застосування, б) із застосуванням вбудованих функцій (визначених у модулі Prelude). Наведіть приклади застосування розроблених функцій для обробки не одного, а кількох наборів вхідних даних.

Варіант: 3

1.3 Визначити частоту кожного елемента списку, напр.: "aaabbcaadddd"⇒ [(’a’,5), (’b’,2), (’c’,1), (’d’,4)].

2.3 Визначити, чи є число простим.

Код програми:

Кожен елемент у списку буде зберігатися як ключ у словнику, а значення ключа буде відповідати кількості входжень цього елемента у список. Для реалізації цього, код використовує модуль Data.Map, який забезпечує ефективну реалізацію словників у Haskell. Коротко про кожну функцію, яка тут використовується:

- frequencyPrelude - це функція, яка бере на вхід список елементів типу a і повертає список пар (a, Int), де кожна пара відповідає ключу та значенню в словнику.
- Map.toList перетворює словник на список пар ключ-значення.
- foldl згортає список від лівого до правого, використовуючи початкове значення порожнього словника Map.empty.
- Map.insertWith (+) x 1 acc додає елемент x до словника acc, збільшуючи його значення на 1, якщо він вже присутній у словнику.

Отже, кожен елемент у списку xs буде доданий до словника Map, а значення його ключа буде збільшене на 1, якщо він вже присутній у словнику. На виході функція повертає список пар ключ-значення словника, який містить частоту кожного елемента у вхідному списку. Для написання функції без Prelude було написано власну імплементацію foldl myFoldl. Функція myFoldl працює рекурсивно і згортає список елементів, починаючи з початкового значення аккумулятора acc. У випадку, якщо список порожній, функція повертає початкове значення аккумулятора. У випадку, якщо список не порожній, функція бере перший елемент списку x та рекурсивно викликає себе зі списком елементів, що залишилися xs і новим значенням аккумулятора f acc x, яке отримано застосуванням функції f до поточного значення аккумулятора acc та першого елементу списку x.

```haskell
import Data.List (sort, group, delete)
import qualified Data.Map.Strict as Map

myFoldl :: (b -> a -> b) -> b -> [a] -> b
myFoldl f acc [] = acc
myFoldl f acc (x:xs) = myFoldl f (f acc x) xs
--task 1
--with prelude (foldl is function defined in Prelude module)
frequencyPrelude :: Ord a => [a] -> [(a, Int)]
frequencyPrelude xs = Map.toList $ foldl (¥acc x -> Map.insertWith (+) x 1 acc) Map.empty
xs

--without prelude
frequencyWithoutPrelude :: Ord a => [a] -> [(a, Int)]
frequencyWithoutPrelude xs = Map.toList $ myFoldl (¥acc x -> Map.insertWith (+) x 1 acc)
Map.empty xs
```

Функція isPrimePrelude перевіряє подільність нашого числа на кожне число з проміжку $x \in [2;\ \lfloor\sqrt{n}\rfloor]$. Для написання аналогічної функції без використання Prelude було написано власні імплементації функціям `mod`, all, а проміжок був збільшений до $[2; n-1]$, щоб не використовувати sqrt та fromIntegral з Prelude.

```haskell
myMod :: Int -> Int -> Int
myMod x y
  | x < y     = x
  | otherwise = myMod (x - y) y

myAll :: (a -> Bool) -> [a] -> Bool
myAll p [] = True
myAll p (x:xs)
  | p x = myAll p xs
  | otherwise = False

--task 2
--with prelude (mod, all, floor, sqrt, fromIntegral are functions defined in Prelude
module)
isPrimePrelude :: Int -> Bool
isPrimePrelude n
  | n <= 1 = False
  | otherwise = all (¥x -> n `mod` x /= 0) [2..limit]
  where limit = floor $ sqrt $ fromIntegral n
--without prelude
isPrimeWithoutPrelude :: Int -> Bool
isPrimeWithoutPrelude n
  | n <= 1 = False
  | otherwise = myAll (¥x ->  myMod n x /= 0) [2..n-1]

main = do
    putStrLn "TASK 1.A: WITH PRELUDE"
    let strP1 = map ((++ "  ") . show) p1
    putStrLn $ concat strP1
    putStrLn "TASK 1.B: WITHOUT PRELUDE"
```

```
    let strWp1 = map ((++ "  ") . show) wp1
    putStrLn $ concat strWp1
    putStrLn "TASK 2.A: WITH PRELUDE"
    let strP2 = map ((++ "  ") . show) p2
    putStrLn $ concat strP2
    putStrLn "TASK 2.B: WITHOUT PRELUDE"
    let strWp2 = map ((++ "  ") . show) wp2
    putStrLn $ concat strWp2
    where
        p1 = map frequencyPrelude arr1
        wp1 = map frequencyWithoutPrelude arr1
        p2 = map isPrimePrelude arr2
        wp2 = map isPrimeWithoutPrelude arr2
        arr1 = ["aaabbcaadddd","22222133333ddd","00022213888833DDDFF"]
        arr2 = [30,31,32,100,101]
```

Результат виконання програми:

```
TASK 1.A: WITH PRELUDE
[('a',5),('b',2),('c',1),('d',4)]
[('1',1),('2',5),('3',5),('d',3)]
[('0',3),('1',1),('2',3),('3',3),('8',4),('D',3),('F',2)]

TASK 1.B: WITHOUT PRELUDE
[('a',5),('b',2),('c',1),('d',4)]
[('1',1),('2',5),('3',5),('d',3)]
[('0',3),('1',1),('2',3),('3',3),('8',4),('D',3),('F',2)]

TASK 2.A: WITH PRELUDE
False
True
False
False
True

TASK 2.B: WITHOUT PRELUDE
False
True
False
False
True
```

Як бачимо, реалізовані функції з Prelude і без нього працюють однаково справно.