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 №1

з дисципліни «Функційне програмування мовою Haskell»

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

Київ – 2023

Тема: визначення рекурсивних функцій.

Мета: набути досвіду визначення рекурсивних функцій, використання механізму зіставлення зі зразком і роботи з кортежами та списками.

Завдання: Визначте вказані функції в кожному з завдань: а) без застосування, б) ззастосуванням вбудованих функцій (визначених у модулі Prelude).

Варіант: 3

1.3 Видалити повтори елементів списку, наприклад: [1,1,1,5,5,3, 1,1,222,222,222,222] ⇒ [1,5,3,222].

2.3 Об'єднання зі змішуванням двох списків довжиною n1 та n2. Вихідний список має довжину 2 · n, де n = $\min(n_1, n_2)$. Наприклад «abcde» та «123» перетворюються на «a1b2c3»

Код програми:

Оскільки в Prelude вже визначена функція, яка видаляє повтори елементів, то ми можемо просто присвоїти її нашій функції. Іншим рішенням буде створити функцію, яка рекурсією проглядає масив на пошук дублікатів. Якщо елемент такий вже є, ми повертаємо рекурсивний виклик нашої функції на хвіст нашого масиву, якщо його немає, то повертаємо конкатенацію голови нашого масиву з рекурсивним викликом нашої функції на хвіст нашого масиву. Для імплементації нашої функції без використання функцій із модуля Prelude можна написати свою реалізацію 'elem' myElem, яка буде рекурсивно визивати себе допоки не найде однаковий елемент, інакше поверне False.

```
import Data.List (intercalate, transpose, nub)
--task 1
--with prelude (elem / nub is function defined in Prelude library)
removeDuplicatesPrelude :: Eq a => [a] -> [a]
-- removeDuplicatesPrelude = nub
removeDuplicatesPrelude [] = []
removeDuplicatesPrelude (x:xs)
    | x `elem` xs = removeDuplicatesPrelude xs
    | otherwise   = x : removeDuplicatesPrelude xs

--without prelude
--custom elem implementation
myElem :: Eq a => a -> [a] -> Bool
myElem _ [] = False
myElem x (y:ys)
    | x == y    = True
    | otherwise = myElem x ys

removeDublicatesWithoutPrelude :: Eq a => [a] -> [a]
removeDublicatesWithoutPrelude [] = []
```

```
removeDublicatesWithoutPrelude (x:xs)
    | myElem x xs = removeDublicatesWithoutPrelude xs
    | otherwise   = x : removeDublicatesWithoutPrelude xs
```

Для реалізації функції merge з використанням Prelude ми спочатку транспонуємо наші масиви, їх розмірність при цьому стане n × 2, тобто n пар масивів розміром 2, який складається з попарних елементів першого і другого рядку, при тому якщо розміри рядків будуть різниться, то елементи, яких бракує, будуть замінені на пусту стрічку при intercalate. Щоб цього уникнути, ми залишаємо лише потрібну кількість елементів за допомогою take (2 · n). Функція intercalate розпаковує другий масив, і кожен його елемент сполучає з першим аргументом (у нашому випадку пустим масивом). У функції без використання Prelude ми оголошуємо клози на випадок, коли нам або першим, або другим аргументом прийде пуста стрічка. Якщо нам передаються не пусті аргументи, то ми конкатенуємо голови наших стрічок і рекурсивний виклик нашої функції до хвостів наших стрічок.

```
--task 2
--with prelude (take and min are functions defined in Prelude)
mergePrelude :: [a] -> [a] -> [a]
mergePrelude [] _ = []
mergePrelude _ [] = []
mergePrelude xs ys = take (2 * n) (intercalate [] (transpose [xs, ys]))
    where n = min (length xs) (length ys)
--without prelude
mergeWithoutPrelude :: [a] -> [a] -> [a]
mergeWithoutPrelude [] _ = []
mergeWithoutPrelude _ [] = []
mergeWithoutPrelude (x:xs) (y:ys) = x:y:mergeWithoutPrelude xs ys

main = do
    putStrLn "TASK 1.A: WITH PRELUDE"
    print p1
    putStrLn "TASK 1.B: WITHOUT PRELUDE"
    print wp1
    putStrLn "TASK 2.A: WITH PRELUDE"
    print p2
    putStrLn "TASK 2.B: WITHOUT PRELUDE"
    print wp2
    where
        p1 = removeDuplicatesPrelude arr0
        wp1 = removeDublicatesWithoutPrelude arr0
        p2 = mergePrelude arr1 arr2
        wp2 = mergeWithoutPrelude arr1 arr2
        arr0 = [1,1,1,5,5,3,  1,1,222,222,222,222]
        arr1 = "abcde"
        arr2 = "123"
```

Результат запуску програми:

```
TASK 1.A: WITH PRELUDE
[5,3,1,222]
TASK 1.B: WITHOUT PRELUDE
[5,3,1,222]
TASK 2.A: WITH PRELUDE
"a1b2c3"
TASK 2.B: WITHOUT PRELUDE
"a1b2c3"
```

Як бачимо, функції з Prelude і без нього дають однаково правильний результат.