Київський національний університет імені Тараса Шевченка

**Лабораторна робота 5**

**З навчального курсу «Парадигми та технології програмування»**

**«Функціональне програмування. Завдання розділу 5.»  
Варіант (8, 4, 0):**

Виконав:

студент 3 курсу

факультету кібернетики

спеціальність «Комп’ютерні науки»

групи ТТП-32

Чебан Богдан Володимирович

**Київ 2024**

**Постановка задачі:**

I. Загальна умова

1) Запрограмувати та протестувати три унарні Maybe-функції для обчислення заданих варіантом трьох виразів, обираючи в останніх в якості константи n Ваш номер у списку з двох академічних груп. (Єдиним аргументом функцій тут вважати x). (2 бали).

2) Запрограмувати із використанням монадних операцій та протестувати Maybe-функцію для обчислення суперпозиції трьох функцій з пункту 1), при тому зовнішньою у суперпозиції має виступати перша з них, внутрішньою – остання:

a) шукану функцію подати із використанням do-нотації. (1 бал).

b) шукану функцію подати без використання do-нотації. (1 бал).

3) Запрограмувати та протестувати бінарну Maybe-функцію для обчислення останнього з трьох заданих варіантом виразу, вважаючи n аргументом, що приймає дійсні значення. (1бал).

4) Запрограмувати із використанням монадних операцій та протестувати Maybe-функцію для обчислення такої суперпозиції, коли у згадану вище (у пункті 3) бінарну функцію замість першого її аргументу підставляється перша з пункту 1) функція, а замість другого аргументу - друга:

a) шукану функцію подати із використанням do-нотації. (1 бал).

b) шукану функцію подати без використання do-нотації. (2 бали).

ВИЗНАЧЕННЯ ВАРІАНТУ:

Мій порядковий номер в списку який ви надали на початку курсу в екселі:

N=40

40\*421= 16840 (звідси беремо останні три цифри (8,4,0)

маємо кортеж виразів(8, 4, 0):

1. 

f= Log10[x-(1/n)]

1. 

f= (1/(Sqrt[x+n]))

1. 

f=Sqrt[x^2-lg(n)]

f= Log10[x-(1/40)]

f= (1/(Sqrt[x+40]))

f=Sqrt[x^2-lg(40)]

**Хід роботи:**

**task 1:**

**import System.IO (hFlush, stdout)**

**import Text.Read (readMaybe)**

**data FunctionType = LogFunction | SqrtFunction1 | SqrtFunction2**

**-- Основна функція**

**main :: IO ()**

**main = do**

**putStrLn "Select function to compute (1 for Log, 2 for Sqrt1, 3 for Sqrt2 or 'Stop' to exit):"**

**hFlush stdout**

**funcTypeInput <- getLine**

**case funcTypeInput of**

**"Stop" -> putStrLn "Program terminated."**

**\_ -> do**

**funcType <- getFunctionType funcTypeInput**

**putStrLn "Enter a number for x (or 'Stop' to exit):"**

**hFlush stdout**

**input <- getLine**

**case input of**

**"Stop" -> putStrLn "Program terminated."**

**\_ -> case readMaybe input of**

**Just x -> do**

**computeAndDisplayResult funcType x**

**main -- repeat the loop**

**Nothing -> do**

**putStrLn "Invalid input, please enter a valid number."**

**main -- repeat the loop**

**getFunctionType :: String -> IO FunctionType**

**getFunctionType input = case input of**

**"1" -> return LogFunction**

**"2" -> return SqrtFunction1**

**"3" -> return SqrtFunction2**

**\_ -> do**

**putStrLn "Invalid input, please enter 1, 2, 3, or 'Stop'."**

**newInput <- getLine**

**getFunctionType newInput**

**computeAndDisplayResult :: (Floating a, Ord a, Show a) => FunctionType -> a -> IO ()**

**computeAndDisplayResult funcType x = do**

**let result = case funcType of**

**LogFunction -> show $ logFunction x**

**SqrtFunction1 -> show $ sqrtFunction1 x**

**SqrtFunction2 -> show $ sqrtFunction2 x**

**putStrLn $ "Result: " ++ result**

**logFunction :: (Floating a, Ord a) => a -> Either String a**

**logFunction x = let value = x - (1 / 40) in**

**if value > 0 then Right (logBase 10 value) else Left "Value under log is non-positive"**

**sqrtFunction1 :: (Floating a, Ord a) => a -> Either String a**

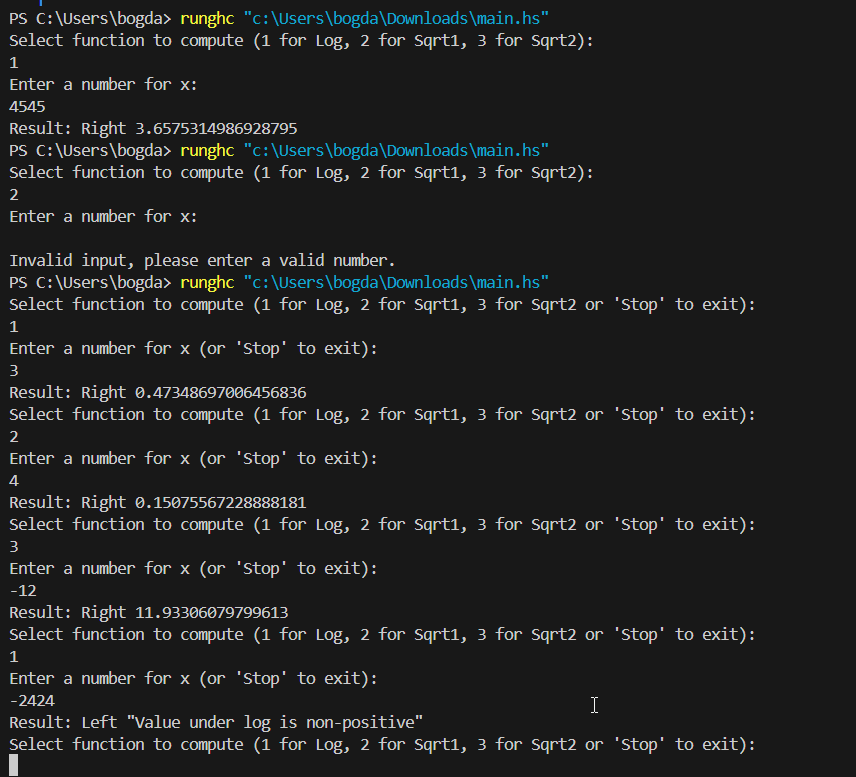
**sqrtFunction1 x = let value = x + 40 in**

**if value >= 0 then Right (1 / sqrt value) else Left "Value under square root is negative"**

**sqrtFunction2 :: (Floating a, Ord a) => a -> Either String a**

**sqrtFunction2 x = let value = x^2 - logBase 10 40 in**

**if value >= 0 then Right (sqrt value) else Left "Value under square root is negative"**

****

**Коротенький опис коду:**

**Типи даних і функції:**

**FunctionType** - перелічувальний тип (enum), який визначає три можливі функції: **LogFunction**, **SqrtFunction1**, **SqrtFunction2**.

**logFunction**, **sqrtFunction1**, **sqrtFunction2** - це функції, які виконують обчислення на основі введеного значення **x**. Вони повертають результат типу **Maybe** або **Either** для обробки помилок або відсутніх значень.

**Основний потік виконання:**

Функція **main** - основний вхід в програму, яка використовує цикл для запиту у користувача яку функцію обчислити або завершити роботу за допомогою команди "Stop".

Використовуються **hFlush stdout** для забезпечення виведення тексту на консоль без буферизації.

**Взаємодія з користувачем:**

Користувач спочатку вибирає тип функції, яку хоче обчислити, вводячи число від 1 до 3, або вводить "Stop" для виходу з програми.

Після вибору функції, користувач вводить число **x**, для якого буде проведено обчислення.

**Обчислення і виведення результатів:**

**computeAndDisplayResult** виконує обрану математичну операцію і виводить результат.

Якщо введення некоректне, користувачеві виводиться відповідне повідомлення про помилку.

**Рекурсивний цикл:**

Програма продовжує запитувати користувача про виконання нових обчислень або завершення роботи до тих пір, поки не буде введено "Stop".

**Task2:**

module Main where

import Text.Read (readMaybe)

t1 :: (Floating a, Ord a) => a -> Maybe a

t1 x = let value = x - (1 / 40) in

if value > 0 then Just (logBase 10 value) else Nothing

t2 :: (Floating a, Ord a) => a -> Maybe a

t2 x = let value = x + 40 in

if value > 0 then Just (1 / sqrt value) else Nothing

t3 :: (Floating a, Ord a) => a -> Maybe a

t3 x = let value = x^2 - logBase 10 40 in

if value >= 0 then Just (sqrt value) else Nothing

-- без do-нотації

composeFunctions :: (Floating a, Ord a) => a -> Maybe a

composeFunctions x = t3 x >>= t2 >>= t1

-- do-нотацією

composeFunctionsDo :: (Floating a, Ord a) => a -> Maybe a

composeFunctionsDo x = do

resultT3 <- t3 x

resultT2 <- t2 resultT3

t1 resultT2

-- Тестова функція, яка виводить результати

testFunction :: (Floating a, Ord a, Show a) => a -> IO ()

testFunction x = do

putStrLn "Result without do-notation:"

print $ composeFunctions x

putStrLn "Result with do-notation:"

print $ composeFunctionsDo x

-- Основна функція

main :: IO ()

main = loop where

loop = do

putStrLn "Task 2: Enter a number for x ('Stop' to exit):"

input <- getLine

case input of

"Stop" -> putStrLn "Program terminated."

\_ -> case readMaybe input of

Just x -> do

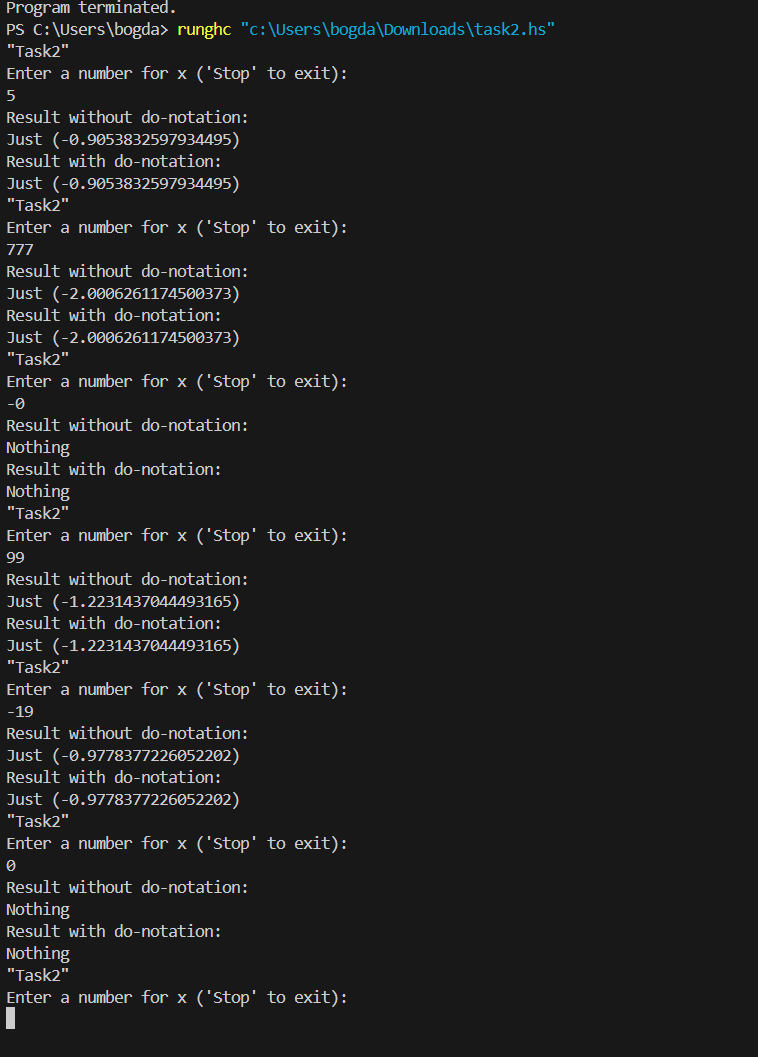
testFunction x

loop -- Repeat the loop

Nothing -> do

putStrLn "Invalid input, please enter a valid number or 'Stop' to exit."

loop -- Repeat the loop

****

**Коротенький опис коду:**

**Функції t1, t2, t3**: Це математичні функції, які повертають **Maybe a**. Вони виконують різні обчислення і повертають **Just** з результатом, якщо обчислення можливе, або **Nothing**, якщо обчислення виконати неможливо (наприклад, через математичні обмеження).

**Функція composeFunctions**: Це функція, що реалізує композицію функцій **t3**, **t2**, **t1** без використання **do**-нотації, використовуючи монадний оператор **>>=** для передачі результатів однієї функції до наступної.

**Функція composeFunctionsDo**: Аналогічно **composeFunctions**, але виконує композицію з використанням **do**-нотації, що робить послідовність обчислень більш читабельною.

**Функція testFunction**: Виводить результати обох композиційних функцій (**composeFunctions** і **composeFunctionsDo**) для заданого введення **x**.

**Функція main**: Це основний вхід у програму, що запускає інтерактивний цикл, де користувач може вводити значення **x** для обчислення, або ввести "Stop" для виходу з програми.

**Task 3:**

Запрограмувати та протестувати бінарну Maybe-функцію для обчислення останнього з трьох заданих варіантом виразу, вважаючи n аргументом, що приймає дійсні значення.

import Text.Read (readMaybe)

-- Функція для обчислення заданої формули

sqrtLogFunction :: (Floating a, Ord a) => a -> a -> Maybe a

sqrtLogFunction x n

| n > 0 = let value = x^2 - logBase 10 n

in if value >= 0 then Just (sqrt value) else Nothing

| otherwise = Nothing

-- Тестова функція для перевірки різних значень

testFunction :: (Floating a, Ord a, Show a) => a -> a -> IO ()

testFunction x n = do

putStrLn $ "Calculating Sqrt[" ++ show x ++ "^2 - log10(" ++ show n ++ ")]:"

print $ sqrtLogFunction x n

-- Основна функція

main :: IO ()

main = loop where

loop = do

putStrLn "Enter a number for x ('Stop' to exit):"

xInput <- getLine

case xInput of

"Stop" -> putStrLn "Program terminated."

\_ -> do

putStrLn "Enter a number for n ('Stop' to exit):"

nInput <- getLine

case nInput of

"Stop" -> putStrLn "Program terminated."

\_ -> case (readMaybe xInput, readMaybe nInput) of

(Just x, Just n) -> do

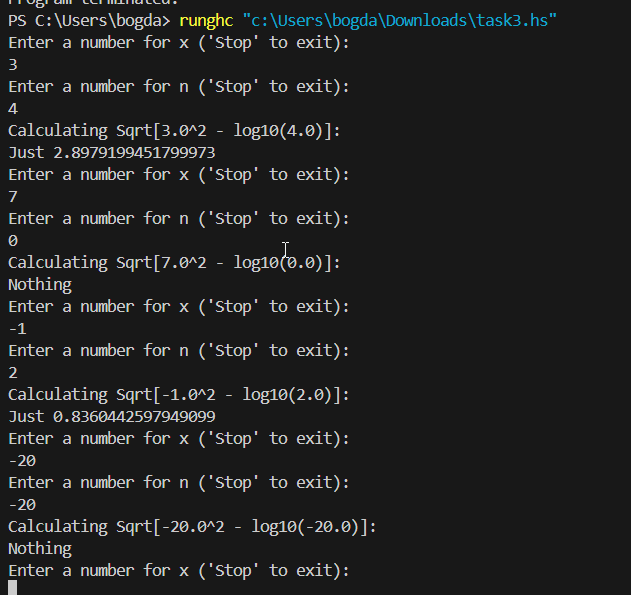
testFunction x n

loop

\_ -> do

putStrLn "Invalid input, please enter valid numbers."

loop



**Коротенький опис коду:**

Цей код має на меті обчислити значення виразу **sqrt(x^2 - logBase 10 n)**, де **x** та **n** - цілі або дійсні числа, які вводяться користувачем. Основна функція **main** використовує цикл введення-виведення, щоб продовжувати запитувати користувача про значення **x** та **n** до тих пір, поки він не введе "Stop".

* Користувач вводить значення **x**.
* Потім він вводить значення **n**.
* Якщо введено "Stop", програма завершується.
* Якщо обидва введення є числами (цілими або дійсними), вони передаються в функцію **testFunction**, яка обчислює вираз **sqrt(x^2 - logBase 10 n)** і виводить результат.
* Якщо введення неправильне (наприклад, не може бути перетворено в число), користувач повторно запитується про введення числа.

Функція **sqrtLogFunction** обчислює заданий вираз **sqrt(x^2 - logBase 10 n)**. Якщо **n** більше 0, то обчислюється значення **value** із застосуванням виразу. Якщо **value** більше або дорівнює 0, то обчислюється квадратний корінь з **value**. Якщо **n** менше або дорівнює 0, функція повертає **Nothing**.

**Task 4:**

import Text.Read (readMaybe)

sqrtLogFunction :: (Floating a, Ord a) => a -> a -> Maybe a

sqrtLogFunction x n

| n > 0 = let value = x^2 - logBase 10 n

in if value >= 0 then Just (sqrt value) else Nothing

| otherwise = Nothing

logFunction :: (Floating a, Ord a) => a -> Maybe a

logFunction x = let value = x - (1 / 40) in

if value > 0 then Just (logBase 10 value) else Nothing

sqrtFunction1 :: (Floating a, Ord a) => a -> Maybe a

sqrtFunction1 x = let value = x + 40 in

if value >= 0 then Just (1 / sqrt value) else Nothing

-- sуперпозиція з do-нотацією

superpositionFunctionDo :: (Floating a, Ord a) => a -> Maybe a

superpositionFunctionDo x = do

xResult <- logFunction x

nResult <- sqrtFunction1 x

sqrtLogFunction xResult nResult

-- sуперпозиція без do-нотації

superpositionFunction :: (Floating a, Ord a) => a -> Maybe a

superpositionFunction x =

logFunction x >>= (\xResult -> sqrtFunction1 x >>= (\nResult -> sqrtLogFunction xResult nResult))

main :: IO ()

main = loop where

loop = do

putStrLn "Enter a number for x ('Stop' to exit):"

input <- getLine

case input of

"Stop" -> putStrLn "Program terminated."

\_ -> case readMaybe input of

Just x -> do

putStrLn "Result with do-notation:"

let resultDo = superpositionFunctionDo x

print resultDo

putStrLn "Result without do-notation:"

let result = superpositionFunction x

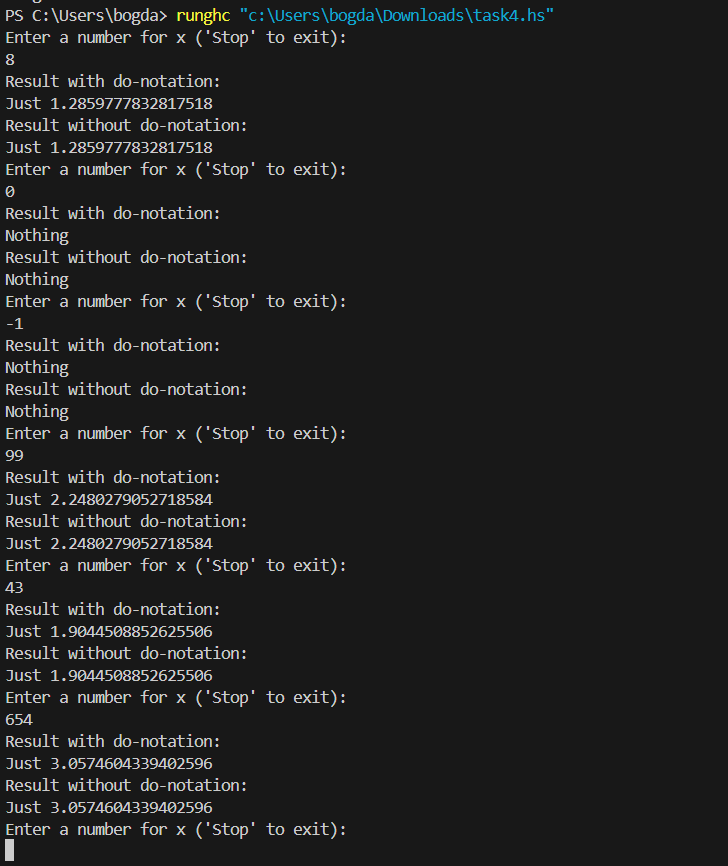
print result

loop

Nothing -> do

putStrLn "Invalid input, please enter a valid number or 'Stop' to exit."

loop



**Коротенький опис коду:**

Цей код об'єднує різні математичні функції і виконує їх композицію, використовуючи як монадичну послідовність з **do**-нотацією, так і без неї.

**Визначення функцій**:

**sqrtLogFunction**: Обчислює значення виразу **sqrt(x^2 - logBase 10 n)**. Якщо **n > 0**, то обчислюється значення **value** із застосуванням виразу, і результат обчислення повертається у формі **Maybe**.

**logFunction**: Обчислює значення логарифму за основою 10 від **x - (1 / 40)**. Якщо **x - (1 / 40) > 0**, то обчислюється логарифм, і результат повертається у формі **Maybe**.

**sqrtFunction1**: Обчислює значення **1 / sqrt(x + 40)**. Якщо **x + 40 >= 0**, то обчислюється ділення одиниці на квадратний корінь, і результат повертається у формі **Maybe**.

**Композиція функцій**:

**superpositionFunctionDo**: Композиція функцій з використанням **do**-нотації. Застосовує послідовність функцій **logFunction**, **sqrtFunction1**, і **sqrtLogFunction** до вхідного значення **x**.

**superpositionFunction**: Композиція функцій без використання **do**-нотації. Використовує оператор **>>=** для послідовної аплікації функцій.

**Основна функція main**:

Запускає цикл, що постійно повторюється, доки користувач не введе "Stop".

Під час кожної ітерації циклу користувачеві пропонується ввести значення **x**.

Після введення **x** викликається **superpositionFunctionDo** та **superpositionFunction** для обчислення результатів композиції функцій і виведення їх на екран.

**Висновок:**

Я навчився працювати з монадами в Haskell, зокрема з Maybe-монадою, для обробки випадків, коли функції можуть не повернути значення. Це особливо корисно при роботі з математичними функціями, що мають обмеження на вхідні дані, як наприклад, невід'ємність підкореневих виразів або область визначення логарифмічних функцій. Я також дізнався, як використовувати do-нотацію та монадні операції, такі як >>= (bind), для створення читабельних і зрозумілих послідовностей обчислень, що включають в себе потенційно відсутні значення.

Крім того, я навчився реалізовувати циклічний ввід від користувача в консольному додатку на Haskell, дозволяючи користувачеві виконувати обчислення неодноразово та завершувати програму за власним вибором. Це знання корисне для створення інтерактивних додатків, які вимагають динамічної взаємодії з користувачем.

В цілому, робота з монадами в Haskell допомогла мені глибше зрозуміти концепції функціонального програмування та покращила мої навички у написанні більш безпечного і виразного коду.