**Розділ 5**

**Виконав Ходаков Максим, ТТП-32**

**Умова**

A close up of a text

Description automatically generated

**Визначення варіанта**

N = 38

(надано у файлі Загальний список студентів та розподіл варіантів.xlsx)

421 \* 38 = 15998, три останні цифри (9, 9, 8)

Модифікуємо кортеж (9, 9, 8) і виходить -> (9, 0, 8).

Отже маємо кортеж виразів (f9, f0, f8)

A black and white math equation

Description automatically generated

A black square with black text

Description automatically generated with medium confidence

A black text on a white background

Description automatically generated

**Завдання 1**

**f9:**

-- Task1  
-- f9  
-- Оголошуємо функцію, яка приймає один аргумент типу Double і повертає Maybe Double  
maybeSqrt :: Double -> Maybe Double  
maybeSqrt x =  
 let value = x - (1/38)  
 in if value < 0 then Nothing else Just (sqrt value)  
  
main :: IO ()  
main = do  
 -- Тестування з додатнім числом, що дає додатній результат під коренем  
 print $ maybeSqrt 2  
 -- Тестування з числом, що призводить до від'ємного результату під коренем  
 print $ maybeSqrt (-1)  
 -- Тестування з межовим значенням (1/38), що має давати 0 під коренем  
 print $ maybeSqrt (1/38)

**A black screen with white text

Description automatically generated**

**f0:**

module Main where  
  
-- Task1  
-- f9  
maybeSqrt :: Double -> Maybe Double  
maybeSqrt x =  
 let value = x - (1/38)  
 in if value < 0 then Nothing else Just (sqrt value)  
  
-- f0  
maybeSqrtExpression :: Double -> Maybe Double  
maybeSqrtExpression x =  
 let value = x^2 - logBase 10 38  
 in if value < 0 then Nothing else Just (sqrt value)  
  
main :: IO ()  
main = do  
 print $ "f9:"  
 -- Тестування з додатнім числом, що дає додатній результат під коренем  
 print $ maybeSqrt 2  
 -- Тестування з числом, що призводить до від'ємного результату під коренем  
 print $ maybeSqrt (-1)  
 -- Тестування з межовим значенням (1/38), що має давати 0 під коренем  
 print $ maybeSqrt (1/38)  
 print $ "f0:"  
 -- Приклад виклику функції з позитивним результатом  
 print $ maybeSqrtExpression 10  
 -- Приклад виклику функції з результатом, який дає від'ємне число під коренем  
 print $ maybeSqrtExpression 1  
 -- Виклик функції зі значенням, що може бути близьким до межі від'ємного під корінним виразом  
 print $ maybeSqrtExpression 2

**A screen shot of a computer

Description automatically generated**

**f8:**

module Main where  
  
-- Task1  
-- f9  
maybeSqrt :: Double -> Maybe Double  
maybeSqrt x =  
 let value = x - (1/38)  
 in if value < 0 then Nothing else Just (sqrt value)  
  
-- f0  
maybeSqrtExpression :: Double -> Maybe Double  
maybeSqrtExpression x =  
 let value = x^2 - logBase 10 38  
 in if value < 0 then Nothing else Just (sqrt value)  
  
-- f8  
maybeLog10 :: Double -> Maybe Double  
maybeLog10 x =  
 let value = x - (1/38)  
 in if value <= 0 then Nothing else Just (logBase 10 value)  
  
main :: IO ()  
main = do  
 print $ "f9:"  
 -- Тестування з додатнім числом, що дає додатній результат під коренем  
 print $ maybeSqrt 2  
 -- Тестування з числом, що призводить до від'ємного результату під коренем  
 print $ maybeSqrt (-1)  
 -- Тестування з межовим значенням (1/38), що має давати 0 під коренем  
 print $ maybeSqrt (1/38)  
 print $ "f0:"  
 -- Приклад виклику функції з позитивним результатом  
 print $ maybeSqrtExpression 10  
 -- Приклад виклику функції з результатом, який дає від'ємне число під коренем  
 print $ maybeSqrtExpression 1  
 -- Виклик функції зі значенням, що може бути близьким до межі від'ємного під корінним виразом  
 print $ maybeSqrtExpression 2  
 putStrLn "f8:"  
 -- Вхідне значення, що призводить до від'ємного аргументу логарифму  
 print $ maybeLog10 1  
 -- Граничне значення, має повернути Nothing  
 print $ maybeLog10 (1/38)  
 -- Позитивне значення, що дає додатній аргумент логарифму  
 print $ maybeLog10 2

**A screenshot of a computer

Description automatically generated**

**Завдання 2**

module Main where  
  
-- Task1  
-- f9  
maybeSqrt :: Double -> Maybe Double  
maybeSqrt x =  
 let value = x - (1/38)  
 in if value < 0 then Nothing else Just (sqrt value)  
  
-- f0  
maybeSqrtExpression :: Double -> Maybe Double  
maybeSqrtExpression x =  
 let value = x^2 - logBase 10 38  
 in if value < 0 then Nothing else Just (sqrt value)  
  
-- f8  
maybeLog10 :: Double -> Maybe Double  
maybeLog10 x =  
 let value = x - (1/38)  
 in if value <= 0 then Nothing else Just (logBase 10 value)  
  
-- Task2  
-- do нотація  
superpositionDo :: Double -> Maybe Double  
superpositionDo x = do  
 result1 <- maybeLog10 x  
 result2 <- maybeSqrtExpression result1  
 maybeSqrt result2  
  
-- без do нотації  
superpositionBind :: Double -> Maybe Double  
superpositionBind x = maybeLog10 x >>= maybeSqrtExpression >>= maybeSqrt  
  
  
main :: IO ()  
main = do  
 print $ "f9:"  
 -- Тестування з додатнім числом, що дає додатній результат під коренем  
 print $ maybeSqrt 2  
 -- Тестування з числом, що призводить до від'ємного результату під коренем  
 print $ maybeSqrt (-1)  
 -- Тестування з межовим значенням (1/38), що має давати 0 під коренем  
 print $ maybeSqrt (1/38)  
 print $ "f0:"  
 -- Приклад виклику функції з позитивним результатом  
 print $ maybeSqrtExpression 10  
 -- Приклад виклику функції з результатом, який дає від'ємне число під коренем  
 print $ maybeSqrtExpression 1  
 -- Виклик функції зі значенням, що може бути близьким до межі від'ємного під корінним виразом  
 print $ maybeSqrtExpression 2  
 putStrLn "f8:"  
 -- Вхідне значення, що призводить до від'ємного аргументу логарифму  
 print $ maybeLog10 1  
 -- Граничне значення, має повернути Nothing  
 print $ maybeLog10 (1/38)  
 -- Позитивне значення, що дає додатній аргумент логарифму  
 print $ maybeLog10 2  
  
 print $ "Task2"  
 -- Позитивний тест, має повернути Just  
 putStrLn "Тест 1 (очікується Just):"  
 print $ superpositionDo 100  
 print $ superpositionBind 100  
  
 -- Тест, який призводить до Nothing на етапі u3  
 putStrLn "\nТест 2 (очікується Nothing на u3):"  
 print $ superpositionDo (1/38)  
 print $ superpositionBind (1/38)  
  
 -- Тест, який може призвести до помилки на етапі u2  
 putStrLn "\nТест 3 (можливе Nothing на u2):"  
 print $ superpositionDo 1  
 print $ superpositionBind 1  
  
 -- Тест, який проходить u3 і u2, але призводить до помилки на u1  
 putStrLn "\nТест 4 (можливе Nothing на u1):"  
 print $ superpositionDo 0.5  
 print $ superpositionBind 0.5

**A screenshot of a computer program

Description automatically generated**

do-нотація в Haskell дозволяє виконувати послідовність дій у монадичному контексті зручнішим та читабельнішим способом. Кожен крок (result1 <- u3 x, result2 <- u2 result1, u1 result2) є монадичною операцією, яка використовує >>= під капотом для передачі результату однієї операції наступній. Якщо будь-яка з функцій u3, u2, чи u1 повертає Nothing, вся послідовність зупиняється і superpositionDo також повертає Nothing. Це типовий приклад використання монадичної операції у Maybe монаді. Оператор >>= (відомий як монадичний бінд) безпосередньо використовується для послідовного застосування функцій до значення, обгорнутого в монаду, де кожна функція може повернути нове значення в монаді Обидва підходи (з використанням do-нотації та >>=) ілюструють застосування монадичних операцій у контексті Maybe монади, дозволяючи ефективно управляти ланцюжками операцій з можливим виникненням помилок або відсутності значень без потреби в явному контролі помилок або розгалуженнях.

**Завдання 3**

-- Task3  
-- Функція для обчислення lg(x - (1/n))  
maybeLog :: Double -> Double -> Maybe Double  
maybeLog n x =  
 let value = x - (1/n)  
 in if value <= 0 then Nothing else Just (logBase 10 value)

print $ "Task3"  
-- Позитивний тест  
putStrLn "Позитивний тест (очікується Just):"  
print $ maybeLog 2 10 -- n=2, x=10  
  
-- Позитивний тест  
putStrLn "Позитивний тест (очікується Just):"  
print $ maybeLog 7.5 10.8  
  
-- Граничний випадок  
putStrLn "\nГраничний випадок (очікується 0.0):"  
print $ maybeLog 2 1.5 -- n=2, x=1.5 (1/2 = 0.5, x - (1/2) = 1)  
  
-- Від'ємний випадок  
putStrLn "\nВід'ємний випадок (очікується Nothing):"  
print $ maybeLog 2 0.5 -- n=2, x=0.5

**A computer screen shot of white text

Description automatically generated**

**Завдання 4**

Спочатку змінимо функцію з пункту 3 щоб вона приймала (x, n) значення а не (n, x) для того щоб відповідати умовам з пояснення.

-- Task3  
-- Функція для обчислення lg(x - (1/n))  
maybeLog :: Double -> Double -> Maybe Double  
maybeLog x n =  
 let value = x - (1/n)  
 in if value <= 0 then Nothing else Just (logBase 10 value)

Змінимо тести

print $ "Task3"  
-- Позитивний тест  
putStrLn "Позитивний тест (очікується Just):"  
print $ maybeLog 10 2 -- n=2, x=10  
  
-- Позитивний тест  
putStrLn "Позитивний тест (очікується Just):"  
print $ maybeLog 10.8 7.5  
  
-- Граничний випадок  
putStrLn "\nГраничний випадок (очікується 0.0):"  
print $ maybeLog 1.5 2 -- n=2, x=1.5 (1/2 = 0.5, x - (1/2) = 1)  
  
-- Від'ємний випадок  
putStrLn "\nВід'ємний випадок (очікується Nothing):"  
print $ maybeLog 0.5 2 -- n=2, x=0.5

A screenshot of a computer program

Description automatically generated

Результати тестів не змінилися можемо будувати функцію з пункту 4

-- Task4  
superpositionDoTask4 :: Double -> Maybe Double  
superpositionDoTask4 x = do  
 firstArg <- maybeSqrt x  
 secondArg <- maybeSqrtExpression x  
 maybeLog firstArg secondArg  
  
superpositionBindTask4 :: Double -> Maybe Double  
superpositionBindTask4 x =  
 maybeSqrt x >>= \firstArg ->  
 maybeSqrtExpression x >>= \secondArg ->  
 maybeLog firstArg secondArg

print $ "Task4"  
-- Тест 1: x = 20, обидва аргументи повернуть Just значення  
putStrLn "Тест 1 (очікується Just):"  
print $ superpositionDoTask4 20  
print $ superpositionBindTask4 20  
  
-- Тест 2: x дуже малий, maybeSqrt поверне Nothing  
putStrLn "\nТест 2 (очікується Nothing через maybeSqrt):"  
print $ superpositionDoTask4 0.02  
print $ superpositionBindTask4 0.02  
  
-- Тест 3: x = 0.5, можливо, що maybeSqrtExpression поверне Nothing  
putStrLn "\nТест 3 (очікується Nothing через maybeSqrtExpression):"  
print $ superpositionDoTask4 0.5  
print $ superpositionBindTask4 0.5  
  
-- Тест 4: x дуже малий, обидва аргументи повернуть Nothing  
putStrLn "\nТест 4 (очікується Nothing, обидва аргументи):"  
print $ superpositionDoTask4 0.001  
print $ superpositionBindTask4 0.001

A screenshot of a computer

Description automatically generated

Використовуються таки самі підходи що і в завданні 2.

**Висновок**

Було реалізовано задачі на математичні вирази використовуючи монадні операції для імплементації суперпозиції.