**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ**

**Лабораторна робота № 4**

з дисципліни «Непроцедурне Програмування »

Тема роботи: «Синтаксис у функціях»

Виконав студент

групи AнД-31

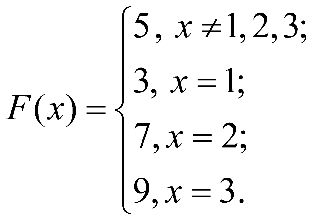
Швед І.Г.

Перевірив:

Гайна Г.А.

**Київ – 2023**

1. **Записати функцію за допомогою варти:**



f :: Int -> Int

f x

    | x == 1 = 3

    | x == 2 = 7

    | x == 3 = 9

    | otherwise = 5

1. **Записати функцію isTriangle, яка перевіряє, чи можна з даних 3 відрізків (довжин) побудувати трикутник.**

isTriangle :: Int -> Int -> Int -> Bool

isTriangle a b c = (a + b > c) && (a + c > b) && (b + c > a)

1. **Функцію, яка шукає мінімум з трьох заданих цілих чисел.**

findMinimum :: Int -> Int -> Int -> Int

findMinimum a b c = min (min a b) c

1. **Функцію, яка перевіряє, чи дані 4 числа задано у порядку спадання.**

areNumbersInDescendingOrder :: Int -> Int -> Int -> Int -> Bool

areNumbersInDescendingOrder a b c d = a >= b && b >= c && c >= d

1. **Функцію, яка приймає координати 3 точок на площині і перевіряє, чи є трикутник, утворений цими 3 точками прямокутним.**

isRightTriangle :: (Int, Int) -> (Int, Int) -> (Int, Int) -> Bool

isRightTriangle (x1, y1) (x2, y2) (x3, y3) =

    let a = distanceSquared (x1, y1) (x2, y2)

        b = distanceSquared (x1, y1) (x3, y3)

        c = distanceSquared (x2, y2) (x3, y3)

    in  a == b + c || b == a + c || c == a + b

  where

    distanceSquared (x1, y1) (x2, y2) = (x2 - x1) ^ 2 + (y2 - y1) ^ 2

1. **Функцію, яка по заданим а і в знаходить роз вязок рівняння   або False, якщо кореня немає.**

solveEquation :: Float -> Float -> Either Float Bool

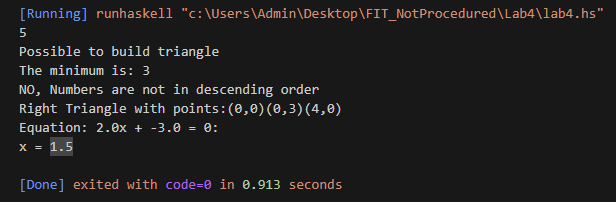
solveEquation a b

     | a /= 0  = Left (-b / a)  -- Решение существует: x = -b / a

     | a == 0 && b /= 0 = Right False  -- Решения не существует

     | a == 0 && b == 0 = Right True  -- Бесконечно много решений

**Результат роботи програми:**

****

**Код програми:**

f :: Int -> Int

f x

    | x == 1 = 3

    | x == 2 = 7

    | x == 3 = 9

    | otherwise = 5

isTriangle :: Int -> Int -> Int -> Bool

isTriangle a b c = (a + b > c) && (a + c > b) && (b + c > a)

findMinimum :: Int -> Int -> Int -> Int

findMinimum a b c = min (min a b) c

areNumbersInDescendingOrder :: Int -> Int -> Int -> Int -> Bool

areNumbersInDescendingOrder a b c d = a >= b && b >= c && c >= d

isRightTriangle :: (Int, Int) -> (Int, Int) -> (Int, Int) -> Bool

isRightTriangle (x1, y1) (x2, y2) (x3, y3) =

    let a = distanceSquared (x1, y1) (x2, y2)

        b = distanceSquared (x1, y1) (x3, y3)

        c = distanceSquared (x2, y2) (x3, y3)

    in  a == b + c || b == a + c || c == a + b

  where

    distanceSquared (x1, y1) (x2, y2) = (x2 - x1) ^ 2 + (y2 - y1) ^ 2

solveEquation :: Float -> Float -> Either Float Bool

solveEquation a b

     | a /= 0  = Left (-b / a)  -- Решение существует: x = -b / a

     | a == 0 && b /= 0 = Right False  -- Решения не существует

     | a == 0 && b == 0 = Right True  -- Бесконечно много решений

main :: IO ()

main = do

    let x = 4

    let result = f x

    print result

    let num1 = 3

    let num2 = 4

    let num3 = 5

    let num4 = 8

    let canFormTriangle = isTriangle num1 num2 num3

    if canFormTriangle

        then putStrLn "Possible to build triangle"

        else putStrLn "NOT possible to build triangle"

    let minimumValue = findMinimum num1 num2 num3

    putStrLn ("The minimum is: " ++ show minimumValue)

    let inDescendingOrder = areNumbersInDescendingOrder num1 num2 num3 num4

    if inDescendingOrder

        then putStrLn "Numbers are in descending order"

        else putStrLn "NO, Numbers are not in descending order"

    let point1 = (0, 0)

    let point2 = (0, 3)

    let point3 = (4, 0)

    let isRight = isRightTriangle point1 point2 point3

    if isRight

        then putStrLn ("Right Triangle with points:" ++show point1 ++show point2 ++show point3)

        else putStrLn ("NOT Right Triangle with points:" ++show point1 ++show point2 ++show point3)

    let a1 = 2

    let b1 = -3

    let result1 = solveEquation a1 b1

    putStrLn ("Equation: " ++ show a1 ++ "x + " ++ show b1 ++ " = 0:")

    case result1 of

        Left x -> putStrLn ("x = " ++ show x)

        Right hasInfiniteSolutions -> putStrLn (if hasInfiniteSolutions then "Eternality roots" else "NO roots at all")

**Висновок:** Протягом виконання лабораторної роботи я навчився працювати з синтаксисом функцій