**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра прикладних інформаційних систем**

# Звіт до лабораторної роботи №3

## з курсу

## «Функціональне програмування»

*Студентки 4 курсу*
*групи ПП-41*
*спеціальності 122 «Комп'ютерні науки»*
*ОП «Прикладне програмування»*
*Штонь Софії Максимівни*

*Викладач:*
Пирог М. В.

**Київ – 2023**

***Тема роботи:*** Замикання (локальні визначення)

***Мета роботи:*** Ознайомитися з особливостями механізму функціонального програмування – замикання. Розширити навички використання анонімних функцій. Поглибити знання в області написання нетривіальних функцій вищого порядку.

## *Теоретичні відомості*

Тема №3 «Функції вищого порядку» має на меті розкриття основних особливостей утворення та використання функцій вищого порядку. Для цього розглядається відображення і згортка, лямбда-вирази та застосування функцій вищого порядку для обробки списків та складних структур. На самостійне опрацювання виноситься питання застосування механізму функціонального програмування – замикання для кращого засвоєння нетривіальних функцій вищого порядку.

Для поглиблення знань в області функцій вищого порядку, особливо при реалізації нетривіальних функцій, слід особливу увагу звернути на анонімні функції взагалі, одним з видів якої на ряду з лямбда-функцією стоїть замикання. Необхідно розглянути особливості локальних визначень та сфери застосування, а також умови використання локальних визначень лише всередині функцій. Крім того, необхідно проаналізувати види замикань та характерні риси кожного з двох видів, а також процес оптимізації обрахунків на основі замикання, що зумовлений можливістю обрахунку замикання лише один раз. Слід визначити схожі риси та різницю між двома видами локальних визначень.

## *Завдання для виконання*

Реалізувати простий лічильник, який зчитує свої виклики та повертає їх поточну кількість використовуючи замикання (локальні визначення) мовою Haskell.

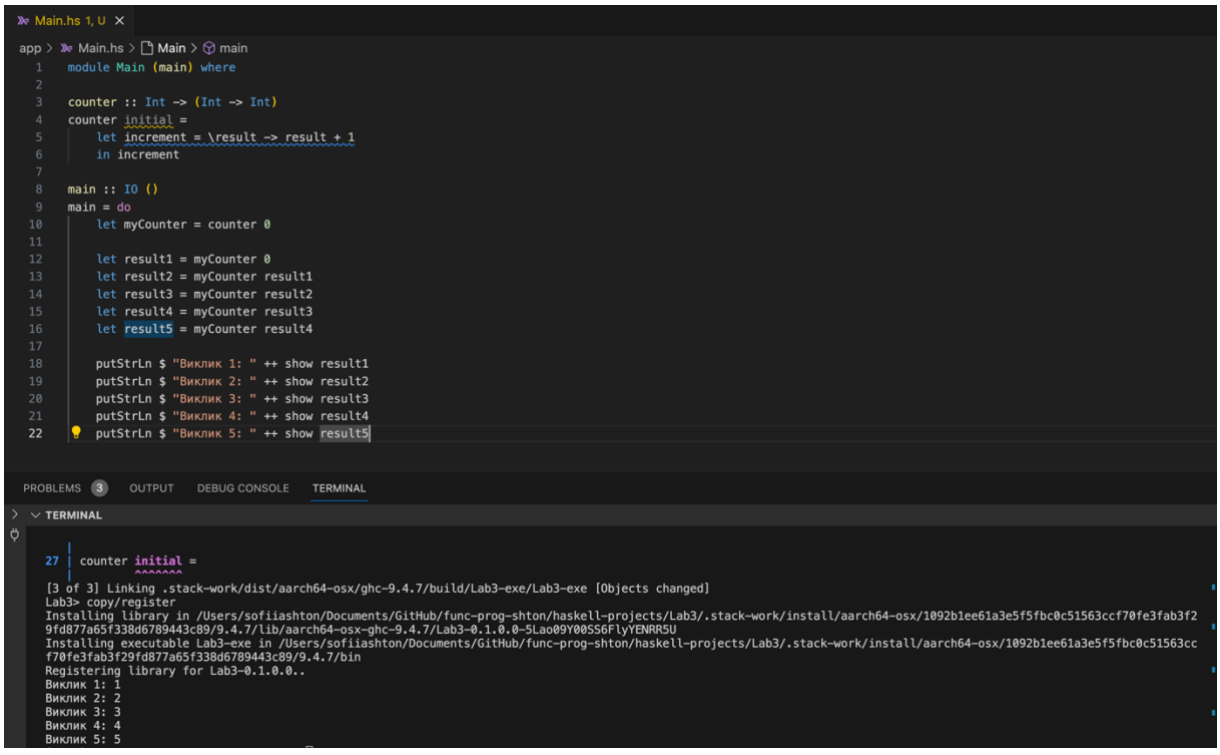
## *Хід роботи*

*Код програми (Haskell):*

```haskell
module Main (main) where


counter :: Int -> (Int -> Int)
counter initial =
  let increment = \result -> result + 1
  in increment
```

```haskell
main :: IO ()
main = do
  let myCounter = counter 0

  let result1 = myCounter 0
  let result2 = myCounter result1
  let result3 = myCounter result2
  let result4 = myCounter result3
  let result5 = myCounter result4

  putStrLn $ "Виклик 1: " ++ show result1
  putStrLn $ "Виклик 2: " ++ show result2
  putStrLn $ "Виклик 3: " ++ show result3
  putStrLn $ "Виклик 4: " ++ show result4
  putStrLn $ "Виклик 5: " ++ show result5
```

*Результат роботи програми:*




***Висновок:*** Отже, у ході цієї лабораторної роботи було проведено ознайомлення з особливостями механізму функціонального програмування – замикання. Було розширено навички використання анонімних функцій та нетривіальних функцій вищого порядку.

### *Контрольні запитання*

*1. Що являє собою замикання (closure) та яким чином воно використовується для оптимізації визначень функцій?*

Замикання — це концепція, яка дозволяє функціям зберігати зовнішні змінні, які не є їхніми параметрами. Коли функція замикається над змінною, вона зберігає не лише значення цієї змінної, але і контекст (оточення), в якому ця змінна була створена. Використання замикань може бути корисним для оптимізації визначень функцій, оскільки це дозволяє уникнути повторного обчислення значень та підтримувати стан між викликами функцій.

*2. Що витікає з твердження, що замикання знаходиться в області імен основної функції?*

Твердження, що замикання знаходиться в області імен основної функції, означає, що функція, яка створює замикання, має доступ до змінних та параметрів області імен, в якій вона сама була визначена. Це важливо для розуміння того, як замикання зберігає контекст. Коли функція замикається над змінною, вона фактично зберігає посилання на цю змінну та всі інші змінні, які знаходяться в тій самій області імен. Таким чином, коли ви викликаєте замкнуту функцію в іншому місці коду, вона все ще може мати доступ до цих змінних, навіть якщо вони не передаються як параметри.

*3. У яких випадках використовується замикання?*

Замикання використовуються для збереження стану функції між викликами, обробки подій, створення анонімних функцій та реалізації функцій вищого порядку.

*4. Яким чином замикання використовується в написання нетривіальних функцій вищого порядку?*

Замикання використовуються для передачі функцій як параметрів, повернення функцій як результатів, часткового застосування параметрів до функцій, збереження стану між викликами тощо.

*5. Які існують види замикання та в чому їх суть?*

- Статичні (Static Closures): Цей тип замикань отримує доступ до змінних з оточення, в якому вони були створені, і зберігає це оточення під час свого створення. Такі замикання залишаються незмінними протягом свого життєвого циклу.
- Динамічні (Dynamic Closures): Ці замикання також отримують доступ до змінних з оточення, але вони можуть оновлюватися, якщо значення змінних змінюється в майбутньому. Вони використовують динамічне оточення для отримання доступу до змінних.