Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчет
по лабораторной работе №3
на тему:

«ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ И ЛЯМБДА-ВЫРАЖЕНИЯ»

Проверил _____ Ф.В. Усенко
(подпись)

Выполнил _____ В.И. Голушко
(подпись) Группа 214301

Минск 2024

Цель работы:

Ознакомиться с основами функционального программирования в Kotlin, изучить лямбда-выражения, анонимные функции и замыкания. Научиться использовать эти концепции для написания более гибкого и читаемого кода.

Задание:

Функция с саморекурсией для работы с деревьями: Напишите программу, которая использует саморекурсивные лямбда-выражения для обхода и обработки бинарного дерева (например, подсчет листьев, поиск максимального значения).

Код программы:

```
data class TreeNode(val value: Int, var left: TreeNode? = null, var right: TreeNode? = null)

fun main() {
    val root = TreeNode(
        10,
        left = TreeNode(5, left = TreeNode(3), right = TreeNode(7)),
        right = TreeNode(7, left = TreeNode(14, left = TreeNode(13), right = TreeNode(14)), right = TreeNode(2))
    )

    lateinit var countLeaves: (TreeNode?) -> Int
    countLeaves = { node ->
        if (node == null) 0
        else if (node.left == null && node.right == null) 1
        else countLeaves(node.left) + countLeaves(node.right)
    }

    lateinit var findMax: (TreeNode?) -> Int?
    findMax = { node ->
        when {
            node == null -> null
            node.right == null && node.left == null -> node.value
            else -> {
                val leftMax = findMax(node.left)
                val rightMax = findMax(node.right)
                maxOf(node.value, leftMax ?: Int.MIN_VALUE, rightMax ?: Int.MIN_VALUE)
            }
        }
    }

    lateinit var findMin: (TreeNode?) -> Int?
    findMin = { node ->
        when {
            node == null -> null
            node.right == null && node.left == null -> node.value
            else -> {
                val leftMax = findMin(node.left)
                val rightMax = findMin(node.right)
```

```
                minOf(node.value,   leftMax   ?:   Int.MIN_VALUE,   rightMax   ?:
Int.MIN_VALUE)
            }
        }
    }

    fun printTree(node: TreeNode?, l: Int){
        if (node!=null) {
            printTree(node.left, l+1)
            for (i in 1..l)
                print("   ")
            println(node.value)
            printTree(node.right, l+1)
        }
    }

    printTree(root, 0)
    println("Количество листьев: ${countLeaves(root)}")
    println("Максимальное значение: ${findMax(root)}")
    println("Минимальное значение: ${findMin(root)}")
}
```

Ответ на контрольные вопросы

1. Как объявить функцию в Kotlin? В чем разница между обычной функцией и однострочной функцией?

Функция объявляется с помощью ключевого слова fun:

```
fun greet(name: String): String {
    return "Hello, $name!"
}
```

Однострочная функция записывается в одну строку:

```
fun greet(name: String) = "Hello, $name!"
```

Разница: однострочная функция не требует явного указания тела {} и оператора return.

2. Как передать функцию в качестве параметра другой функции? Приведите пример.

Функцию можно передать как параметр, задав тип (параметры) -> результат:

```
fun operate(a: Int, b: Int, operation: (Int, Int) -> Int): Int {
    return operation(a, b)
}

val sum = operate(5, 3) { x, y -> x + y }
```

```
println(sum) // 8
```

3. Как создать лямбда-выражение с двумя параметрами? Приведите пример.

Лямбда с двумя параметрами создаётся следующим образом:

```
val multiply: (Int, Int) -> Int = { a, b -> a * b }
println(multiply(4, 5)) // 20
```

4. Как функция reduce работает с массивами? Приведите пример использования.

Функция reduce последовательно применяет операцию к элементам массива, начиная с первого:

```
val numbers = listOf(1, 2, 3, 4)
val sum = numbers.reduce { acc, num -> acc + num }
println(sum) // 10
```

Здесь acc — накопитель, который содержит промежуточный результат.

5. Как создать замыкание, которое увеличивает значение переменной-счетчика при каждом вызове?

Замыкание создаётся через функцию, которая возвращает другую функцию:

```
fun makeCounter(): () -> Int {
    var count = 0
    return { ++count }
}

val counter = makeCounter()
println(counter()) // 1
println(counter()) // 2
```

Вывод:

Ознакомился с основами функционального программирования в Kotlin, изучил работу с лямбда-выражения, анонимные функции и замыкания. Закрепил знания о лямбда-выражениях, написанием кода для выполнения задания повариантy.