Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

Информатики и радиоэлектроники»

Учебная дисциплина: «Программирование мобильных информационных систем»

Отчет

по выполнению лабораторной работы «Основы языка Kotlin»

Выполнил студент: гр. 210101 Разуванов Ф.В.

Проверил: Усенко Ф.В.

Минск 2024

**1. Цель работы:** изучить основные конструкции языка Kotlin, научиться работать с типами данных, операциями ввода-вывода, а также условными операторами. Закрепить полученные знания через выполнение задач, требующих практического применения теории.

**2. Задание:** Напишите программу, которая позволяет пользователю вводить выражения с переменными (например, "x + y \* 2") и сохранять переменные для последующего использования. Программа должна поддерживать операции присваивания и выводить результат вычисления.

**3. Оснащение работы:** техническое задание, технический проект, ЭВМ,

import java.util.\*

fun main() {

val variables = mutableMapOf<String, Double>()

println("Введите выражение с переменными (например, x + y \* 2) или присваивание (например, x = 10). Для выхода напишите 'exit'.")

while (true) {

print("> ")

val input = readLine()?.trim()

if (input == "exit") {

println("Выход.")

break

}

if (input != null) {

try {

if (input.contains("=")) {

handleAssignment(input, variables)

} else {

val variableNames = extractVariables(input)

for (variable in variableNames) {

if (!variables.containsKey(variable)) {

print("Введите значение для переменной $variable: ")

val value = readLine()?.toDoubleOrNull()

if (value != null) {

variables[variable] = value

} else {

throw Exception("Неверное значение для переменной $variable")

}

}

}

val result = evaluateExpression(input, variables)

println("Результат: $result")

}

} catch (e: Exception) {

println("Ошибка: ${e.message}")

}

}

}

}

fun handleAssignment(input: String, variables: MutableMap<String, Double>) {

val parts = input.split("=")

if (parts.size == 2) {

val variable = parts[0].trim()

val expression = parts[1].trim()

if (variable.matches(Regex("[a-zA-Z]+"))) {

val value = evaluateExpression(expression, variables)

variables[variable] = value

println("Переменная $variable установлена в $value")

} else {

throw Exception("Неверное имя переменной. Используйте латинские буквы для имен переменных.")

}

} else {

throw Exception("Неверный синтаксис присваивания. Пример: x = 5")

}

}

fun extractVariables(expression: String): Set<String> {

val variableRegex = Regex("[a-zA-Z]+")

return variableRegex.findAll(expression).map { it.value }.toSet()

}

fun evaluateExpression(expression: String, variables: Map<String, Double>): Double {

val tokens = tokenizeExpression(expression, variables)

return calculateRPN(tokens)

}

fun tokenizeExpression(expression: String, variables: Map<String, Double>): List<String> {

val output = mutableListOf<String>()

val operators = Stack<String>()

val tokens = expression.split(Regex("(?<=op)|(?=op)".replace("op", "[-+\*/^()]")))

val precedence = mapOf("+" to 1, "-" to 1, "\*" to 2, "/" to 2, "^" to 3)

for (token in tokens) {

val trimmedToken = token.trim()

when {

// используется для сопоставления чисел

trimmedToken.matches(Regex("-?\\d+(\\.\\d+)?")) -> {

output.add(trimmedToken)

}

trimmedToken.matches(Regex("[a-zA-Z]+")) -> {

val value = variables[trimmedToken] ?: throw Exception("Неопределенная переменная $trimmedToken")

output.add(value.toString())

}

trimmedToken == "(" -> {

operators.push(trimmedToken)

}

trimmedToken == ")" -> {

while (operators.isNotEmpty() && operators.peek() != "(") {

output.add(operators.pop())

}

operators.pop()

}

precedence.containsKey(trimmedToken) -> {

while (operators.isNotEmpty() && precedence[operators.peek()] ?: 0 > precedence[trimmedToken]!!) {

output.add(operators.pop())

}

operators.push(trimmedToken)

}

}

}

while (operators.isNotEmpty()) {

output.add(operators.pop())

}

println(output)

return output

}

fun calculateRPN(tokens: List<String>): Double {

val stack = Stack<Double>()

for (token in tokens) {

when {

token.matches(Regex("-?\\d+(\\.\\d+)?")) -> {

stack.push(token.toDouble())

}

token in listOf("+", "-", "\*", "/", "^") -> {

val b = stack.pop()

val a = stack.pop()

stack.push(when (token) {

"+" -> a + b

"-" -> a - b

"\*" -> a \* b

"/" -> a / b

"^" -> Math.pow(a, b)

else -> throw Exception("Неверный оператор")

})

}

}

}

return stack.pop()

}

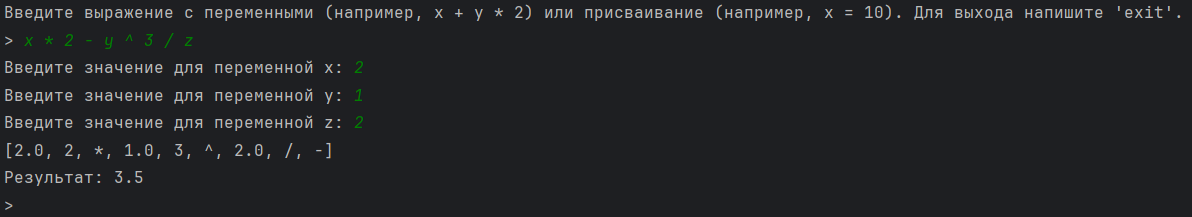


Рисунок 2 – Результат исполнения кода программы