Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Отчет

по практической работе №1

на тему:

**ОСНОВЫ ЯЗЫКА KOTLIN**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Усенко Ф.В.

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Папко К.А.

(подпись)  гр. 214301

Минск, 2024

**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.

1. ВЫПОЛНЕНИЕ ЗАДАНИЯ

import java.util.Stack  
  
class ExpressionSolver {  
 private val variables = mutableMapOf<String, Double>()  
 private val expressions = mutableListOf<String>()  
  
 fun evaluate(input: String) {  
 if (input.contains("=")) {  
 val parts = input.split("=")  
 val left = parts[0].trim()  
 val right = parts[1].trim()  
  
 if (isSingleVariable(left)) {  
 val variable = if (isSingleVariable(left)) left else right  
 val value = if (isSingleVariable(left)) evaluateExpression(right) else evaluateExpression(left)  
  
 if (value != null) {  
 val isNew = variables.put(variable, value) == null  
 println(if (isNew) "Создана новая переменная $variable = $value" else "Переменная $variable обновлена до $value")  
 processPendingExpressions()  
 } else {  
 println("Невозможно вычислить значение для $input")  
 }  
 } else {  
 expressions.add(input)  
 if (!solveExpression(left, right)) solveExpression(right, left)  
 }  
 } else {  
 processSimpleExpression(input)  
 }  
 }  
  
 private fun processPendingExpressions() {  
 val iterator = expressions.iterator()  
 while (iterator.hasNext()) {  
 val expr = iterator.next()  
 val parts = expr.split("=")  
 val left = parts[0].trim()  
 val right = parts[1].trim()  
 if (solveExpression(left, right) || solveExpression(right, left)) {  
 iterator.remove()  
 }  
 }  
 }  
  
 private fun processSimpleExpression(expression: String) {  
 val unknowns = findUnknownVariables(expression)  
 if (unknowns.isEmpty()) {  
 val result = evaluateExpression(expression)  
 if (result != null) {  
 println("Результат выражения $expression = $result")  
 }  
 } else {  
 println("Неизвестные переменные: ${unknowns.joinToString(", ")}")  
 }  
 }  
  
 private fun isSingleVariable(expr: String) = expr.matches(Regex("[a-zA-Z]+"))  
  
 private fun solveExpression(left: String, right: String): Boolean {  
 val knownVariables = variables.keys  
 val unknowns = findUnknownVariables(left)  
  
 return when {  
 unknowns.size > 1 -> {  
 println("Неизвестные переменные: ${unknowns.joinToString(", ")}")  
 false  
 }  
 unknowns.size == 1 -> {  
 val unknown = unknowns.first()  
 val result = evaluateExpression(right) ?: return false  
 val expressionWithKnownValues = left.replace(unknown, "0")  
 val evaluatedWithKnowns = evaluateExpression(expressionWithKnownValues)  
  
 if (evaluatedWithKnowns != null) {  
 val unknownCoefficient = evaluateExpression(left.replace(unknown, "1"))!! - evaluatedWithKnowns  
 if (unknownCoefficient != 0.0) {  
 val unknownValue = (result - evaluatedWithKnowns) / unknownCoefficient  
 variables[unknown] = unknownValue  
 println("Найдена новая переменная $unknown = $unknownValue")  
 return true  
 }  
 }  
 false  
 }  
 else -> {  
 val leftValue = evaluateExpression(left)  
 val rightValue = evaluateExpression(right)  
 if (leftValue != null && rightValue != null) {  
 println(if (leftValue == rightValue) "Верное утверждение" else "Неверное утверждение")  
 true  
 } else {  
 false  
 }  
 }  
 }  
 }  
  
 private fun findUnknownVariables(expression: String): Set<String> {  
 return expression.split(Regex("[+\\-\*/()]"))  
 .map **{** it.trim() **}** .filter **{** it.isNotEmpty() && it.matches(Regex("[a-zA-Z]+")) && it !in variables.keys **}** .toSet()  
 }  
  
 private fun evaluateExpression(expression: String): Double? {  
 try {  
 val tokens = parseTokens(expression)  
 val postfix = infixToPostfix(tokens)  
 return evaluatePostfix(postfix)  
 } catch (e: Exception) {  
 return null  
 }  
 }  
  
 private fun parseTokens(expression: String): List<String> {  
 val regex = Regex("[a-zA-Z]+|\\d+(\\.\\d+)?|[+\\-\*/()]")  
 return regex.findAll(expression).map **{** it.value **}**.toList()  
 }  
  
 private fun infixToPostfix(tokens: List<String>): List<String> {  
 val precedence = mapOf("+" to 1, "-" to 1, "\*" to 2, "/" to 2)  
 val output = mutableListOf<String>()  
 val operators = Stack<String>()  
  
 for (token in tokens) {  
 when {  
 token.matches(Regex("\\d+(\\.\\d+)?")) || token.matches(Regex("[a-zA-Z]+")) -> output.add(token)  
 token == "(" -> operators.push(token)  
 token == ")" -> {  
 while (operators.isNotEmpty() && operators.peek() != "(") {  
 output.add(operators.pop())  
 }  
 if (operators.isNotEmpty()) operators.pop()  
 }  
 token in precedence -> {  
 while (operators.isNotEmpty() && operators.peek() != "(" && precedence[operators.peek()]!! >= precedence[token]!!) {  
 output.add(operators.pop())  
 }  
 operators.push(token)  
 }  
 }  
 }  
 while (operators.isNotEmpty()) output.add(operators.pop())  
 return output  
 }  
  
 private fun evaluatePostfix(postfix: List<String>): Double? {  
 val stack = Stack<Double>()  
  
 for (token in postfix) {  
 when {  
 token.matches(Regex("\\d+(\\.\\d+)?")) -> stack.push(token.toDouble())  
 token.matches(Regex("[a-zA-Z]+")) -> stack.push(variables[token] ?: return null)  
 token == "+" -> stack.push(stack.pop() + stack.pop())  
 token == "-" -> {  
 val b = stack.pop()  
 stack.push(stack.pop() - b)  
 }  
 token == "\*" -> stack.push(stack.pop() \* stack.pop())  
 token == "/" -> {  
 val b = stack.pop()  
 stack.push(stack.pop() / b)  
 }  
 }  
 }  
 return if (stack.size == 1) stack.pop() else null  
 }  
}  
  
fun main() {  
 val solver = ExpressionSolver()  
 println("Введите выражение или 'exit' для выхода:")  
  
 while (true) {  
 print("> ")  
 val input = readLine() ?: break  
 if (input.lowercase() == "exit") {  
 println("Выход из программы.")  
 break  
 }  
 solver.evaluate(input)  
 }  
}

1. ОТВЕТЫ НА ВОПРОСЫ
2. **Что такое функция main в Kotlin, и какова её роль в программе?**  
   Функция main — это точка входа программы в Kotlin, с которой начинается выполнение кода.
3. **Чем отличаются переменные, объявленные с использованием val и var?**  
   Переменные, объявленные с помощью val, нельзя изменить после инициализации, а переменные, объявленные с использованием var, можно изменить.
4. **Какие типы данных существуют в Kotlin для целых чисел?**  
   Для целых чисел в Kotlin используются типы Byte, Short, Int, и Long.
5. **Как объявить строковую переменную в Kotlin? Можно ли её изменить после объявления?**  
   Строковую переменную можно объявить с использованием типов val или var. Изменить её можно только если она объявлена с помощью var.
6. **Что такое интерполяция строк, и как она работает в Kotlin?**  
   Интерполяция строк позволяет включать значения переменных или результаты выражений прямо в строку с помощью символа $.
7. ВЫВОД

В ходе выполнения практической работы был изучен язык программирования Kotlin и его основы. Были рассмотрены ключевые концепции, такие как объявление переменных, типы данных, строковые операции, а также интерполяция строк.