ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Н. И. Чулочникова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3

КАЛЬКУЛЯТОР

по курсу:

КРОССПЛАТФОРМЕННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1 Цель работы

Цель работы: разработать мобильное приложение-калькулятор для ОС Android на языке Kotlin, реализовав интуитивно понятный интерфейс и корректное выполнение математических вычислений.

2 Задание

Работа выполнялась по варианту № 17 (2).

Разработайте калькулятор для арифметических операций в соответствии с вариантом (тригонометрический калькулятор). Интерфейс должен быть интуитивно понятным, результат вычислений должен выводиться пользователю в Activity, все цифры и математические операции должны иметь отдельные кнопки.

3 Листинг программы

Листинг программного кода приложения представлен в Приложении А.

4 Результаты работы программы

На рисунках 1–3 изображены результаты сборки и запуска программы.

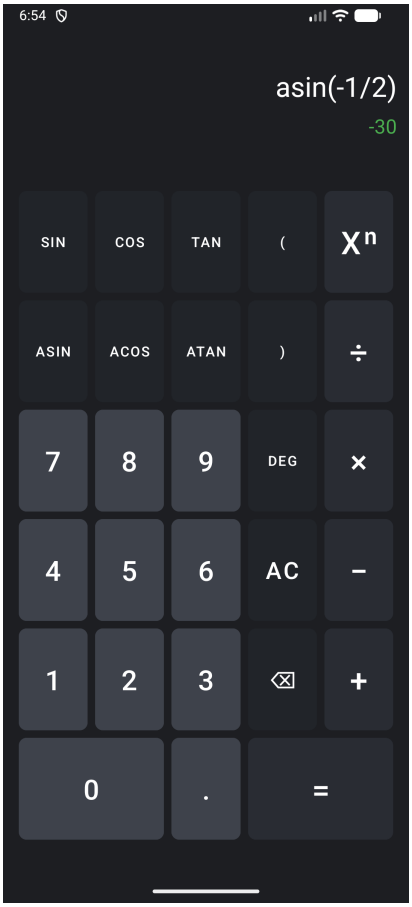

Рисунок 1 — Тестовый расчет 1

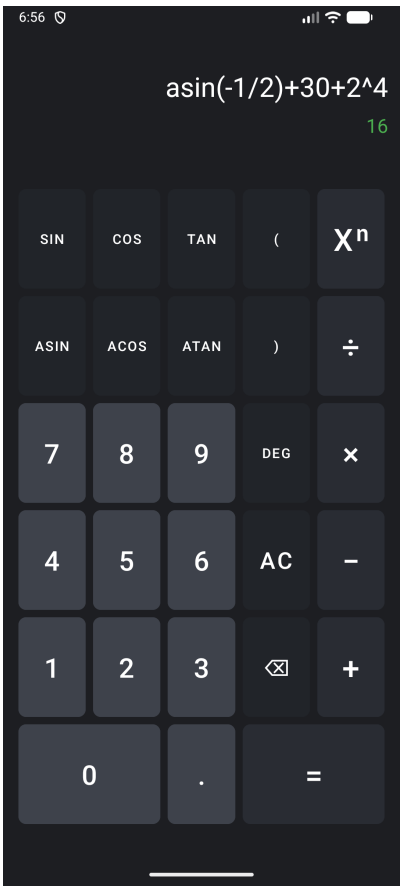

Рисунок 2 — Тестовый расчет 2

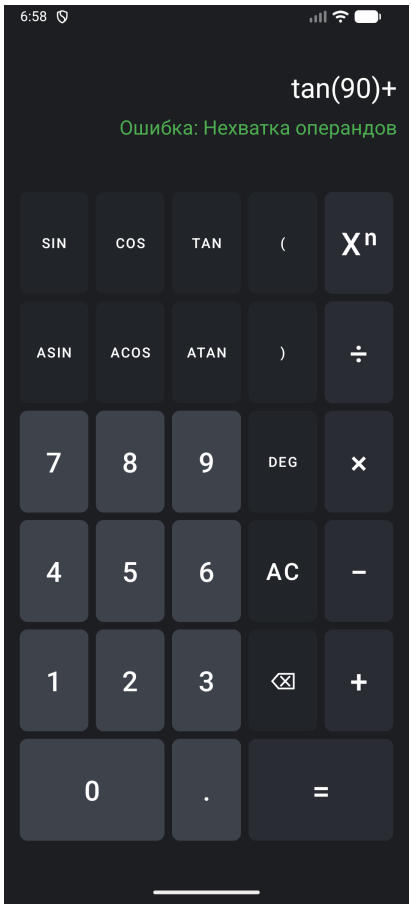

Рисунок 3 — Попытка расчета неполного выражения привела к ошибке

5 Выводы

Разработанное мобильное приложение представляет собой тригонометрический калькулятор, реализованный на языке программирования Kotlin под операционную систему Android. В ходе выполнения работы был создан интуитивно понятный интерфейс, включающий отдельные кнопки для цифр, базовых математических операций и тригонометрических функций. Отображение результатов вычислений организовано в основной Activity, что обеспечивает удобство взаимодействия пользователя с приложением.

В процессе разработки были изучены и применены элементы архитектуры Android-приложений: структура проекта, работа с layout-макетами интерфейса, использование виджетов, обработка событий нажатий кнопок, а также механизм связи логики приложения с элементами пользовательского интерфейса.

Работа также позволила закрепить навыки преобразования строковых данных, применения математических функций Kotlin, обработки пользовательского ввода и формирования корректных выражений для вычислений. Реализована поддержка тригонометрических операций, возведения в степень, работы с радианами и другими функциями, что делает приложение функциональным и полезным.

В результате выполнения задания был получен полностью работоспособный прототип мобильного тригонометрического калькулятора, который может служить основой для дальнейшего расширения: добавления научных функций, истории вычислений и других возможностей. Работа показала важность понимания принципов построения Android-интерфейсов и взаимодействия между компонентами приложения.

ПРИЛОЖЕНИЕ А

Листинг 1 — MainActivity.kt

```
package com.grigorijtomczuk.trigcalculator

import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import java.util.Locale
import java.util.Stack
import kotlin.math.acos
import kotlin.math.asin
import kotlin.math.atan
import kotlin.math.cos
import kotlin.math.pow
import kotlin.math.sin
import kotlin.math.tan

// Главная активность приложения-калькулятора
class MainActivity : AppCompatActivity() {

    // Элементы интерфейса для ввода и отображения результата
    private lateinit var tvInput: TextView
    private lateinit var tvResult: TextView

    // Флаг для переключения между градусами и радианами
    private var degreeMode = true // По умолчанию градусы (DEG)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Инициализация текстовых полей
        tvInput = findViewById(R.id.tvInput)
        tvResult = findViewById(R.id.tvResult)

        // Настройка обработчиков нажатий на кнопки

        // Утилитарные кнопки
        // Кнопка переключения между градусами и радианами
        findViewById<Button>(R.id.btnDegRad).apply {
            text = if (degreeMode) "DEG" else "RAD"
            setOnClickListener {
                degreeMode = !degreeMode
                text = if (degreeMode) "DEG" else "RAD"
            }
        }
        findViewById<Button>(R.id.btnClear).setOnClickListener { clearAll() } //
Очистить все
        findViewById<Button>(R.id.btnDel).setOnClickListener { backspace() }   //
Стереть последний символ
        findViewById<Button>(R.id.btnOpen).setOnClickListener { appendToInput("(") }
// Открывающая скобка
        findViewById<Button>(R.id.btnClose).setOnClickListener { appendToInput(")") }
// Закрывающая скобка

        // Цифры и точка
        val btnIds = listOf(
            R.id.btn0, R.id.btn1, R.id.btn2, R.id.btn3, R.id.btn4,
```

```
            R.id.btn5, R.id.btn6, R.id.btn7, R.id.btn8, R.id.btn9, R.id.btnDot
        )
        val btnTexts = listOf("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ".")
        for (i in btnIds.indices) {
            findViewById<Button>(btnIds[i]).setOnClickListener {
appendToInput(btnTexts[i]) }
        }

        // Операции
        findViewById<Button>(R.id.btnPlus).setOnClickListener { appendToInput("+") }
        findViewById<Button>(R.id.btnSub).setOnClickListener { appendToInput("-") }
        findViewById<Button>(R.id.btnMul).setOnClickListener { appendToInput("*") }
        findViewById<Button>(R.id.btnDiv).setOnClickListener { appendToInput("/") }
        findViewById<Button>(R.id.btnPow).setOnClickListener { appendToInput("^") }

        // Тригонометрические функции
        findViewById<Button>(R.id.btnSin).setOnClickListener { appendToInput("sin(")
}
        findViewById<Button>(R.id.btnCos).setOnClickListener { appendToInput("cos(")
}
        findViewById<Button>(R.id.btnTan).setOnClickListener { appendToInput("tan(")
}
        findViewById<Button>(R.id.btnAsin).setOnClickListener {
appendToInput("asin(") }
        findViewById<Button>(R.id.btnAcos).setOnClickListener {
appendToInput("acos(") }
        findViewById<Button>(R.id.btnAten).setOnClickListener {
appendToInput("atan(") }

        // Кнопка "равно" для вычисления выражения
        findViewById<Button>(R.id.btnEqual).setOnClickListener { evaluateExpression()
}
    }

    // Добавляет строку к текущему вводу
    private fun appendToInput(s: String) {
        tvInput.text = tvInput.text.toString() + s
    }

    // Очищает поля ввода и результата
    private fun clearAll() {
        tvInput.text = ""
        tvResult.text = ""
    }

    // Удаляет последний символ из поля ввода
    private fun backspace() {
        val txt = tvInput.text.toString()
        if (txt.isNotEmpty()) tvInput.text = txt.substring(0, txt.length - 1)
    }

    // Вычисляет выражение, введенное пользователем
    private fun evaluateExpression() {
        val expr = tvInput.text.toString()
        if (expr.isBlank()) return // Ничего не делать, если поле ввода пустое
        try {
            val rpn = infixToRPN(expr) // Преобразовать инфиксную нотацию в обратную
польскую (RPN)
            val value = evalRPN(rpn) // Вычислить RPN-выражение
            tvResult.text = formatDouble(value) // Отобразить результат
        } catch (e: Exception) {
            tvResult.text = "Ошибка: ${e.message}"
```

```
        }
    }

    // Форматирует Double в строку для отображения
    private fun formatDouble(v: Double): String {
        if (v.isNaN()) return "NaN"
        if (v.isInfinite()) return if (v > 0) "Infinity" else "-Infinity"
        val longVal = v.toLong()
        // Если число целое, показать как Long, иначе - как Double с 10 знаками после
запятой
        return if (v == longVal.toDouble()) longVal.toString() else String.format(
            Locale.US,
            "%.10f",
            v
        ).trimEnd('0').trimEnd('.')
    }

    // Парсер: Алгоритм "сортировочной станции" для преобразования в RPN (обратную
польскую нотацию)
    // Поддерживает: числа, операторы (+ - * / ^), скобки, функции (sin, cos, tan,
asin, acos, atan)
    private fun infixToRPN(expr: String): List<String> {
        val output = ArrayList<String>() // Выходная очередь токенов в RPN
        val ops = Stack<String>() // Стек для операторов и функций

        val tokens = tokenize(expr) // Разбиение строки на токены
        var i = 0
        while (i < tokens.size) {
            val token = tokens[i]
            when {
                token.isNumber() -> output.add(token) // Если токен - число, добавить
в выход
                token.isFunction() -> ops.push(token) // Если токен - функция,
положить в стек
                token == "," -> {
                    // Разделитель аргументов функции (в данном калькуляторе не
используется, но оставлен для расширяемости)
                    while (ops.isNotEmpty() && ops.peek() != "(") {
                        output.add(ops.pop())
                    }
                }

                token.isOperator() -> {
                    // Пока на вершине стека оператор с большим или равным
приоритетом (и лево-ассоциативный)
                    while (ops.isNotEmpty() && ops.peek().isOperator() &&
                        ((token.isLeftAssoc() && token.precedence() <=
ops.peek().precedence()) ||
                                (!token.isLeftAssoc() && token.precedence() <
ops.peek()
                                    .precedence()))
                    ) {
                        output.add(ops.pop())
                    }
                    ops.push(token) // Положить текущий оператор в стек
                }

                token == "(" -> ops.push(token) // Открывающая скобка всегда кладется
в стек
                token == ")" -> {
                    // Переместить операторы из стека в выход до открывающей скобки
                    while (ops.isNotEmpty() && ops.peek() != "(")
```

```
output.add(ops.pop())
                    if (ops.isEmpty()) throw Exception("Непарная )") // Ошибка, если
стек пуст
                    ops.pop() // Выкинуть "("
                    // Если после скобки была функция, переместить ее в выход
                    if (ops.isNotEmpty() && ops.peek().isFunction())
output.add(ops.pop())
                }

                else -> throw Exception("Неизвестный токен: $token")
            }
            i++
        }
        // Переместить оставшиеся операторы из стека в выход
        while (ops.isNotEmpty()) {
            val op = ops.pop()
            if (op == "(" || op == ")") throw Exception("Непарная скобка") // Ошибка,
если в стеке остались скобки
            output.add(op)
        }
        return output
    }

    // Вычисляет выражение в обратной польской нотации (RPN)
    private fun evalRPN(tokens: List<String>): Double {
        val st = Stack<Double>() // Стек для чисел
        for (t in tokens) {
            when {
                t.isNumber() -> st.push(t.toDouble()) // Если токен - число, положить
в стек
                t.isOperator() -> {
                    val b = st.popOrError() // Взять второй операнд
                    // Унарный минус требует только один операнд
                    val a = if (t != "u-") st.popOrError() else 0.0
                    val res = when (t) {
                        "+" -> a + b
                        "-" -> a - b
                        "*" -> a * b
                        "/" -> {
                            if (b == 0.0) throw Exception("Деление на ноль")
                            a / b
                        }

                        "^" -> a.pow(b)
                        "u-" -> -b // Обработка унарного минуса
                        else -> throw Exception("Неподдерживаемая операция $t")
                    }
                    st.push(res) // Положить результат обратно в стек
                }

                t.isFunction() -> {
                    val arg = st.popOrError() // Взять аргумент функции из стека
                    val res = when (t.lowercase(Locale.getDefault())) {
                        "sin" -> trigSin(arg)
                        "cos" -> trigCos(arg)
                        "tan" -> trigTan(arg)
                        "asin" -> trigAsin(arg)
                        "acos" -> trigAcos(arg)
                        "atan" -> trigAtan(arg)
                        else -> throw Exception("Неизвестная функция $t")
                    }
                    st.push(res) // Положить результат обратно в стек
```

```
                }

                else -> throw Exception("Неожиданный токен в RPN: $t")
            }
        }
        if (st.size != 1) throw Exception("Неверное выражение") // В конце в стеке
должно остаться одно число
        return st.pop()
    }

    // Функции-обертки для тригонометрии с учетом режима DEG/RAD
    // Для прямых функций: конвертируем градусы в радианы, если нужно
    private fun trigSin(x: Double): Double = sin(if (degreeMode) Math.toRadians(x)
else x)
    private fun trigCos(x: Double): Double = cos(if (degreeMode) Math.toRadians(x)
else x)
    private fun trigTan(x: Double): Double = tan(if (degreeMode) Math.toRadians(x)
else x)

    // Для обратных функций: конвертируем результат в градусы, если нужно
    private fun trigAsin(x: Double): Double {
        val r = asin(x)
        return if (degreeMode) Math.toDegrees(r) else r
    }

    private fun trigAcos(x: Double): Double {
        val r = acos(x)
        return if (degreeMode) Math.toDegrees(r) else r
    }

    private fun trigAtan(x: Double): Double {
        val r = atan(x)
        return if (degreeMode) Math.toDegrees(r) else r
    }

    // Токенизатор: разбивает строку на список токенов (числа, операторы, функции,
скобки)
    private fun tokenize(s: String): List<String> {
        val out = ArrayList<String>()
        var i = 0
        // Удаляем все пробелы из строки
        val str = s.replace("\\s+".toRegex(), "")
        while (i < str.length) {
            val ch = str[i]
            when {
                // Если символ - цифра или точка, считываем все число
                ch.isDigit() || ch == '.' -> {
                    val sb = StringBuilder()
                    while (i < str.length && (str[i].isDigit() || str[i] == '.')) {
                        sb.append(str[i]); i++
                    }
                    out.add(sb.toString())
                    continue // Пропускаем инкремент i в конце цикла
                }
                // Если символ - буква, считываем всю функцию
                ch.isLetter() -> {
                    val sb = StringBuilder()
                    while (i < str.length && str[i].isLetter()) {
                        sb.append(str[i]); i++
                    }
                    out.add(sb.toString())
                    continue // Пропускаем инкремент i
```

```
                }
                // Если символ - оператор
                ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^' -> {
                    val token = ch.toString()
                    // Обработка унарного минуса: если он в начале, после оператора
или '('
                    if (ch == '-') {
                        val prev = out.lastOrNull()
                        if (prev == null || prev in listOf("(", "+", "-", "*", "/",
"^", ",")) {
                            out.add("u-") // Специальный токен для унарного минуса
                        } else {
                            out.add(token) // Бинарный минус
                        }
                    } else out.add(token)
                }
                // Скобки и запятые добавляются как отдельные токены
                ch == '(' || ch == ')' || ch == ',' -> out.add(ch.toString())
                else -> throw Exception("Недопустимый символ '${ch}'")
            }
            i++
        }
        return out
    }

    // Вспомогательные функции и расширения

    // Проверяет, является ли строка числом
    private fun String.isNumber(): Boolean = this.matches(Regex("^[0-9]*\\.?[0-
9]+$"))

    // Проверяет, является ли строка оператором
    private fun String.isOperator(): Boolean = this in listOf("+", "-", "*", "/",
"^", "u-")

    // Проверяет, является ли строка функцией
    private fun String.isFunction(): Boolean {
        val f = this.lowercase(Locale.getDefault())
        return f in listOf("sin", "cos", "tan", "asin", "acos", "atan")
    }

    // Возвращает приоритет оператора
    private fun String.precedence(): Int = when (this) {
        "u-" -> 5 // Унарный минус имеет самый высокий приоритет
        "^" -> 4  // Возведение в степень
        "*", "/" -> 3
        "+", "-" -> 2
        else -> 0
    }

    // Проверяет, является ли оператор лево-ассоциативным
    private fun String.isLeftAssoc(): Boolean = this != "^" && this != "u-"

    // Безопасно извлекает элемент из стека Double, иначе выбрасывает исключение
    private fun Stack<Double>.popOrError(): Double {
        if (this.isEmpty()) throw Exception("Нехватка операндов")
        return this.pop()
    }
}
```