CODE:

package com.example.lab2  
  
import android.os.Bundle  
import android.widget.Toast  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.enableEdgeToEdge  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.tooling.preview.Preview  
import androidx.compose.ui.unit.dp  
import com.example.lab2.ui.theme.Lab2Theme  
import kotlinx.coroutines.launch  
import java.util.\*  
  
class MainActivity : ComponentActivity() {  
 @OptIn(ExperimentalMaterial3Api::class)  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 *setContent* **{** Lab2Theme **{** Scaffold(  
 modifier = Modifier.*fillMaxSize*(),  
 topBar = **{** TopAppBar(  
 title = **{** Text("Calculator") **}** )  
 **}** ) **{** paddingValues **->** Box(modifier = Modifier.*padding*(paddingValues)) **{** Calculator()  
 **}  
 }  
  
 }  
 }** }  
}  
  
@Composable  
fun Calculator() {  
 var expression by remember **{** *mutableStateOf*("") **}** val context = *LocalContext*.current  
 val snackbarHostState = remember **{** SnackbarHostState() **}** val scope = rememberCoroutineScope()  
  
 fun evaluateExpression(expr: String): String {  
 return try {  
 val result = *evaluate*(expr)  
 result.toString()  
 } catch (e: Exception) {  
 scope.*launch* **{** snackbarHostState.showSnackbar(  
 message = "Invalid Expression",  
 actionLabel = "Dismiss",  
 duration = SnackbarDuration.*Indefinite* )  
 **}** ""  
  
 }  
 }  
  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(16.*dp*),  
 verticalArrangement = Arrangement.Bottom,  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** TextField(  
 value = expression,  
 onValueChange = **{** expression = **it }**,  
 modifier = Modifier.*fillMaxWidth*(),  
 readOnly = true,  
 textStyle = *LocalTextStyle*.current.copy(textAlign = TextAlign.End)  
 )  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 SnackbarHost(hostState = snackbarHostState)  
  
 val buttons = *listOf*(  
 *listOf*("C", "DEL", "%", "/"),  
 *listOf*("7", "8", "9", "\*"),  
 *listOf*("4", "5", "6", "-"),  
 *listOf*("1", "2", "3", "+"),  
 *listOf*("00", "0", ".", "=")  
 )  
  
 buttons.*forEach* **{** row **->** Row(modifier = Modifier.*fillMaxWidth*(), horizontalArrangement = Arrangement.SpaceEvenly) **{** row.*forEach* **{** label **->** if (label.*isNotEmpty*()) {  
 FilledTonalButton(  
 onClick = **{** when (label) {  
 "C" -> expression = ""  
 "DEL" -> if (expression.*isNotEmpty*()) expression = expression.*dropLast*(1)  
 "=" -> expression = evaluateExpression(expression)  
 else -> expression += label  
 }  
 **}**,  
 modifier = Modifier  
 .*height*(100.*dp*)  
 .*width*(100.*dp*)  
 .*padding*(2.*dp*)  
 ) **{** Text(label)  
 **}** } else {  
 Spacer(modifier = Modifier.*weight*(1f))  
 }  
 **}  
 }  
 }  
 }**}  
  
// mathematical expressions  
fun evaluate(expr: String): Double {  
 val tokens = expr.*replace*(" ", "").*toCharArray*()  
 val values = Stack<Double>()  
 val ops = Stack<Char>()  
  
 var i = 0  
 while (i < tokens.size) {  
 when {  
 tokens[i].*isDigit*() -> {  
 val sb = StringBuilder()  
 while (i < tokens.size && (tokens[i].*isDigit*() || tokens[i] == '.')) {  
 sb.append(tokens[i])  
 i++  
 }  
 values.push(sb.toString().*toDouble*())  
 i--  
 }  
 tokens[i] in *listOf*('+', '-', '\*', '/') -> {  
 while (ops.*isNotEmpty*() && *precedence*(ops.peek()) >= *precedence*(tokens[i])) {  
 values.push(*applyOp*(ops.pop(), values.pop(), values.pop()))  
 }  
 ops.push(tokens[i])  
 }  
 }  
 i++  
 }  
  
 while (ops.*isNotEmpty*()) {  
 values.push(*applyOp*(ops.pop(), values.pop(), values.pop()))  
 }  
  
 return if (values.*isNotEmpty*()) values.pop() else 0.0  
}  
  
// operator precedence  
fun precedence(op: Char): Int {  
 return when (op) {  
 '+', '-' -> 1  
 '\*', '/' -> 2  
 else -> -1  
 }  
}  
  
fun applyOp(op: Char, b: Double, a: Double): Double {  
 return when (op) {  
 '+' -> a + b  
 '-' -> a - b  
 '\*' -> a \* b  
 '/' -> if (b != 0.0) a / b else throw ArithmeticException("Division by zero")  
 else -> 0.0  
 }  
}  
  
@Preview(showBackground = true)  
@Composable  
fun CalculatorPreview() {  
 Lab2Theme **{** Calculator()  
 **}**}

OUTPUT:



