Assignment No :B3

1 Aim

Generate a scientific calculator.

2 Problem Statement

A mobile application needs to be designed for using a Calculator (+, - ,*, /, Sin, Cos, sq-root) with Memory Save/Recall using Extended precision floating point number format.
Give the Required modeling, Design and Positive-Negative test cases.

3 Learning Objectives

1. To study mobile development and applications.
2. To implement scientific calculator operations.
3. To learn android programming.

4 Learning Outcome

1. Learn java programming for android application.
2. To be able to create an android application.

5 Mathematical Model

Let S be the system that represents the above problem statement.
Initially,
S ={ S_t, E_t, I, O, F_{me}, DD, NDD, S_c, F_c }
Where,
S ={ϕ}
E = Calculated output

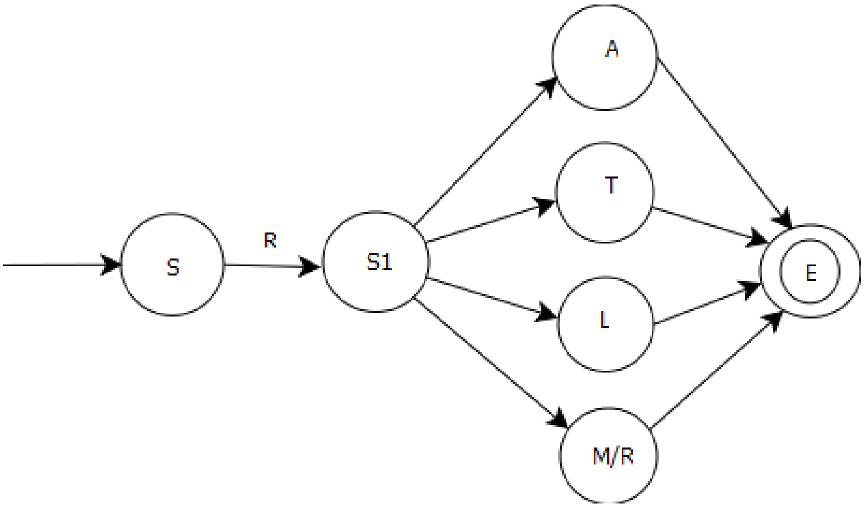
I = R { set of real numbers}
O = R { set of real numbers}
$F_m e$ = {F_1, F_2 F_A, F_T, F_M, F_R, F_L } - Set of main functions.
where,
F_1 = insert number
$\phi \longrightarrow$ R
F_2 = display output
R $\longrightarrow$ R
F_A = { F_{add}, F_{sub}, F_{mul}, F_{div} } - Arithmetic functions.
F_T = { F_{sin}, F_{cos}, F_{tan} } - Trigonometric functions.
F_M = { F_{m+} } - Memory functions.
F_R = { F_{mc} } - Memory recall functions.
F_L = { F_{log10}, F_{loge} } - Logarithmic functions.

S_c (Success case) = response after button click
correct output

F_c (Failure case) = app failed to respond
incorrect output

DD (Deterministic data) = F_{me}
NDD (Non deterministic data) = O

6 State Diagram



```
\item s - start state
\item s1 - input numbers
\item A - arithmetic functions.
\item T - trigonometric functions.
\item L - logarithmic functions.
\item M/R- memory/recall functions.
\item E - display calculated output.
```

7 Theory

7.1 Building android application

Event Handling

Events are a useful way to collect data about a userâs interaction with interactive components of your app, like button presses or screen touch etc. The Android framework maintains an event queue into which events are placed as they occur and then each event is removed from the queue on a first-in, first-out (FIFO) basis. One can capture these events in program and take appropriate action as per requirements. There are following three concepts related to Android Event Management:

Event Listeners:
The View class is mainly involved in building up a Android GUI, same View class provides a number of Event Listeners. The Event Listener is the object that receives notification when an event happens.

Event Listeners Registration:
Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

Event Handlers:
When an event happens and have registered the event, the event listener calls the Event Handlers, which is the method that actually handles the event.
Example:
1. onClick():

OnClickListener() is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. It uses onClick() event handler to handle such event.

Java Math Class

The java.lang.Math class contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Math.pow():
The java.lang.Math.pow(double a, double b) returns the value of the first argument raised to the power of the second argument.

Math.tan():
The java.lang.Math.tan(double a) returns the trigonometric tangent of an angle.

Math.cos():
The java.lang.Math.cos(double a) returns the trigonometric cosine of an angle.

Math.sin():
The java.lang.Math.sin(double a) returns the trigonometric sine of an angle.

Math.sqrt():
The java.lang.Math.sqrt(double a) returns the correctly rounded positive square root of a double value.

Math.log():
The java.lang.Math.log(double a) returns the natural logarithm (base e) of a double value.

Math.log10():
The java.lang.Math.log(double a) returns the common logarithm (base 10) of a double value.

7.2 Parallelism and Scaling

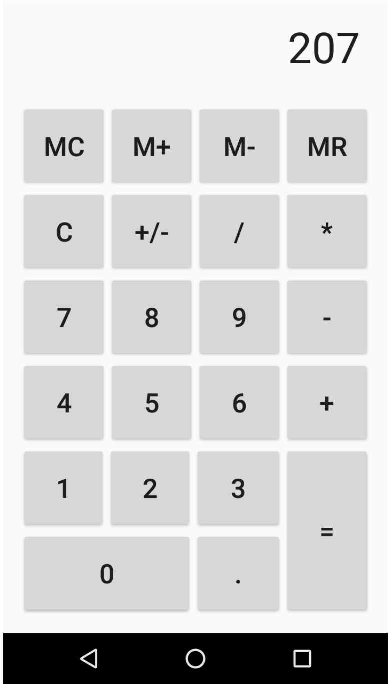
The current setup works well enough on small amounts of data, but at some point data sets can grow sufficiently large that a single computer cannot hold all of the data at once, or the tree becomes so large that it takes an unreasonable amount of time to complete a traversal. To overcome these issues, parallel computing, or parallelism, can be employed. In parallel computing, a problem is divided up and each of multiple processors performs a part of the problem. The processors work at the same time, in parallel.In a tree, each node/subtree is independent. As a result, we can split up a large tree into 2, 4, 8, or more subtrees and hold subtree on each processor. Then, the only duplicated data that must be kept on all processors is the tiny tip of the tree that is the parent of all of the individual subtrees.

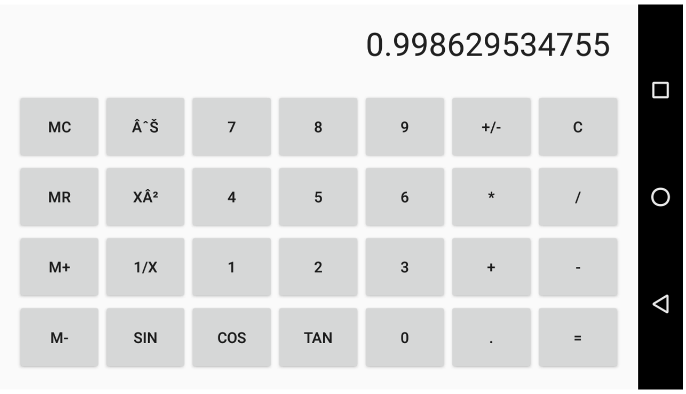
Mathematically speaking, for a tree divided among n processors (where n is a power of two), the processors only need to hold n-1 nodes in common , no matter how big the tree itself is. Throughout the module, a processors rank is an identifier to keep it distinct from the other processors.

8. Source Code

```
//MainActivity.java
package com.example.nsk.calculator;

import java.text.DecimalFormat;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListen

    private TextView mCalculatorDisplay;
    private Boolean userIsInTheMiddleOfTypingANumber = false;
private CalculatorBrain mCalculatorBrain;
private static final String DIGITS = "0123456789.";

DecimalFormat df = new DecimalFormat("@###########");

@SuppressLint("NewApi")
@Override
protected void onCreate(Bundle savedInstanceState) {

    // hide the window title.
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    // hide the status bar and other OS-level chrome
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSC

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mCalculatorBrain = new CalculatorBrain();
    mCalculatorDisplay = (TextView) findViewById(R.id.textView1

    df.setMinimumFractionDigits(0);
    df.setMinimumIntegerDigits(1);
    df.setMaximumIntegerDigits(8);

    findViewById(R.id.button0).setOnClickListener(this);
    findViewById(R.id.button1).setOnClickListener(this);
    findViewById(R.id.button2).setOnClickListener(this);
    findViewById(R.id.button3).setOnClickListener(this);
    findViewById(R.id.button4).setOnClickListener(this);
    findViewById(R.id.button5).setOnClickListener(this);
    findViewById(R.id.button6).setOnClickListener(this);
    findViewById(R.id.button7).setOnClickListener(this);
    findViewById(R.id.button8).setOnClickListener(this);
    findViewById(R.id.button9).setOnClickListener(this);

    findViewById(R.id.buttonAdd).setOnClickListener(this);
    findViewById(R.id.buttonSubtract).setOnClickListener(this);
    findViewById(R.id.buttonMultiply).setOnClickListener(this);
    findViewById(R.id.buttonDivide).setOnClickListener(this);
    findViewById(R.id.buttonToggleSign).setOnClickListener(this
    findViewById(R.id.buttonDecimalPoint).setOnClickListener(th
    findViewById(R.id.buttonEquals).setOnClickListener(this);
    findViewById(R.id.buttonClear).setOnClickListener(this);
    findViewById(R.id.buttonClearMemory).setOnClickListener(thi
    findViewById(R.id.buttonAddToMemory).setOnClickListener(thi
    findViewById(R.id.buttonSubtractFromMemory).setOnClickListe
    findViewById(R.id.buttonRecallMemory).setOnClickListener(th

    // The following buttons only exist in layout-land (Landsca
    // The messier option is to place the buttons in the regula
    if (findViewById(R.id.buttonSquareRoot) != null) {
        findViewById(R.id.buttonSquareRoot).setOnClickListener(
    }
    if (findViewById(R.id.buttonSquared) != null) {
        findViewById(R.id.buttonSquared).setOnClickListener(thi
    }
    if (findViewById(R.id.buttonInvert) != null) {
        findViewById(R.id.buttonInvert).setOnClickListener(this
    }
    if (findViewById(R.id.buttonSine) != null) {
        findViewById(R.id.buttonSine).setOnClickListener(this);
    }
    if (findViewById(R.id.buttonCosine) != null) {
        findViewById(R.id.buttonCosine).setOnClickListener(this
    }
    if (findViewById(R.id.buttonTangent) != null) {
        findViewById(R.id.buttonTangent).setOnClickListener(thi
    }
}

@Override
public void onClick(View v) {

    String buttonPressed = ((Button) v).getText().toString();

    if (DIGITS.contains(buttonPressed)) {

        // digit was pressed
        if (userIsInTheMiddleOfTypingANumber) {

            if (buttonPressed.equals(".") && mCalculatorDisplay
                // ERROR PREVENTION
                // Eliminate entering multiple decimals
            } else {
                mCalculatorDisplay.append(buttonPressed);
            }
        } else {

            if (buttonPressed.equals(".")) {
                // ERROR PREVENTION
                // This will avoid error if only the decimal is
                // before the decimal
                mCalculatorDisplay.setText(0 + buttonPressed);
            } else {
                mCalculatorDisplay.setText(buttonPressed);
            }

            userIsInTheMiddleOfTypingANumber = true;
        }

    } else {
        // operation was pressed
        if (userIsInTheMiddleOfTypingANumber) {

            mCalculatorBrain.setOperand(Double.parseDouble(mCa
            userIsInTheMiddleOfTypingANumber = false;
                            }
                    mCalculatorBrain.performOperation(buttonPre
        mCalculatorDisplay.setText(df.format(mCalculatorBrain.g
            }
    }
    @Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // Save variables on screen orientation change
    outState.putDouble("OPERAND", mCalculatorBrain.getResult())
    outState.putDouble("MEMORY", mCalculatorBrain.getMemory());
}
    @Override
protected void onRestoreInstanceState(Bundle savedInstanceState
    super.onRestoreInstanceState(savedInstanceState);
    // Restore variables on screen orientation change
    mCalculatorBrain.setOperand(savedInstanceState.getDouble("O
    mCalculatorBrain.setMemory(savedInstanceState.getDouble("ME
    mCalculatorDisplay.setText(df.format(mCalculatorBrain.getRe
}}
```

8 Output





9 Conclusion

Thus, we have successfully generated a scientific calculator application using android programming. It contains different mathematical operations such as arithmetic, trigonometric, exponential, logarithmic and memory-recall functions. Java programming is used to perform the same.