# NOISE POLLUTION MONITORING USING IOT

## PHASE 4: DEVELOPMENT PART -2:

### C PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to read noise data from a sensor
double readNoiseData() {
    // Implement code to read data from the noise sensor
    // Return the noise level as a double value
    return 0.0;
}

// Function to analyze the noise data
int analyzeNoiseData(double noiseLevel) {
    // Implement code to analyze the noise level
    // Return a status code indicating the analysis result
    if (noiseLevel > 70.0) {
        return 1;  // Noise pollution detected
    } else {
        return 0;  // Noise pollution within acceptable levels
    }
}

int main() {
    double noiseLevel;
    int status;

    while (1) {
        // Read noise data from the sensor
        noiseLevel = readNoiseData();

        // Analyze the noise data
        status = analyzeNoiseData(noiseLevel);

        // Get the current timestamp
        time_t t = time(NULL);
        struct tm tm = *localtime(&t);

        // Print the result and timestamp
        if (status == 1) {
            printf("Noise pollution detected at %d-%02d-%02d %02d:%02d:%02d\n",
                   tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
                   tm.tm_hour, tm.tm_min, tm.tm_sec);
            // Implement code for sending alerts or notifications here
        } else {
            printf("Noise level within acceptable limits at %d-%02d-%02d %02d:%02d:%02d\n",
                   tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
              tm.tm_hour, tm.tm_min, tm.tm_sec);
      }

      // Sleep for a defined interval before taking the next reading
      // Adjust the interval as needed
      sleep(60);  // Sleep for 60 seconds (1 minute)
   }

   return 0;
}
```

## C++ PROGRAM:

```
#include <iostream>
#include <ctime>
#include "NoiseSensorLibrary.h"  // Include your sensor library here

// Function to read noise data from a sensor
double readNoiseData() {
   // Implement code to read data from the noise sensor
   // You should use functions provided by your sensor library
   double noiseLevel = NoiseSensor::readNoiseLevel();
   return noiseLevel;
}

// Function to analyze the noise data
bool analyzeNoiseData(double noiseLevel) {
   // Implement code to analyze the noise level
   // You can define thresholds and criteria specific to your application
   return (noiseLevel > 70.0);  // Adjust the threshold as needed
}

int main() {
   double noiseLevel;

   while (true) {
      // Read noise data from the sensor
      noiseLevel = readNoiseData();

      // Analyze the noise data
      bool pollutionDetected = analyzeNoiseData(noiseLevel);

      // Get the current timestamp
      std::time_t currentTime = std::time(nullptr);
      std::tm* localTime = std::localtime(&currentTime);

      // Print the result and timestamp
      if (pollutionDetected) {
         std::cout << "Noise pollution detected at "
                 << (localTime->tm_year + 1900) << '-'
                 << (localTime->tm_mon + 1) << '-'
                 << localTime->tm_mday << ' '
                 << localTime->tm_hour << ':'
                 << localTime->tm_min << ':'
                 << localTime->tm_sec << std::endl;
```

```
            // Implement code for sending alerts or notifications here
        } else {
            std::cout << "Noise level within acceptable limits at "
                      << (localTime->tm_year + 1900) << '-'
                      << (localTime->tm_mon + 1) << '-'
                      << localTime->tm_mday << ' '
                      << localTime->tm_hour << ':'
                      << localTime->tm_min << ':'
                      << localTime->tm_sec << std::endl;
        }

        // Sleep for a defined interval before taking the next reading
        // Adjust the interval as needed
        std::this_thread::sleep_for(std::chrono::minutes(1));  // Sleep for 1 minute
    }

    return 0;
 }
```

## MICROPROCESSOR PROGRAM:

```
#include <avr/io.h>
#include
<avr/interrupt.h>


#define F_CPU 16000000UL // CPU clock frequency (16
MHz) #define BAUD 9600     // Baud rate for serial
communication


void USART_Init(unsigned int ubrr) {

   // Set baud rate

   UBRR0H = (unsigned char)(ubrr >>
   8); UBRR0L = (unsigned char)ubrr;


   // Enable receiver and transmitter
   UCSR0B = (1 << RXEN0) | (1 <<
   TXEN0);


   // Set frame format: 8 data, 1 stop bit
   UCSR0C = (1 << UCSZ00) | (1 <<
   UCSZ01);
}
```

```
 void USART_Transmit(unsigned char data) {

   // Wait for empty transmit buffer
   while (!(UCSR0A & (1 << UDRE0)));

// Put data into buffer and send
UDR0 = data;
 }



 int main(void) {

   // Initialize USART
   communication
   USART_Init(F_CPU / 16 / BAUD -
   1);


   // Initialize ADC

   ADMUX = (1 << REFS0); // Reference voltage to AVCC

   ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) | (1 << ADPS2) | (1 <<
   ADPS1);
 // Enable ADC, start conversion, enable ADC interrupt, and set prescaler



   sei(); // Enable global interrupts



   while (1) {

     // Main loop

   }



   return 0;

 }



 ISR(ADC_vect) {

   // ADC conversion complete interrupt



   uint8_t lowByte = ADCL;
   uint8_t highByte =
   ADCH;
   uint16_t adcValue = (highByte << 8) | lowByte
// You can perform noise analysis here and set thresholds for pollution monitoring
```

```
    // Print the ADC value to the serial
    communication USART_Transmit(lowByte);
    USART_Transmit(highByte);
    USART_Transmit('\n');



    // Start the next ADC
    conversion ADCSRA |= (1 <<
    ADSC);
}
```

## HTML PROGRAM:

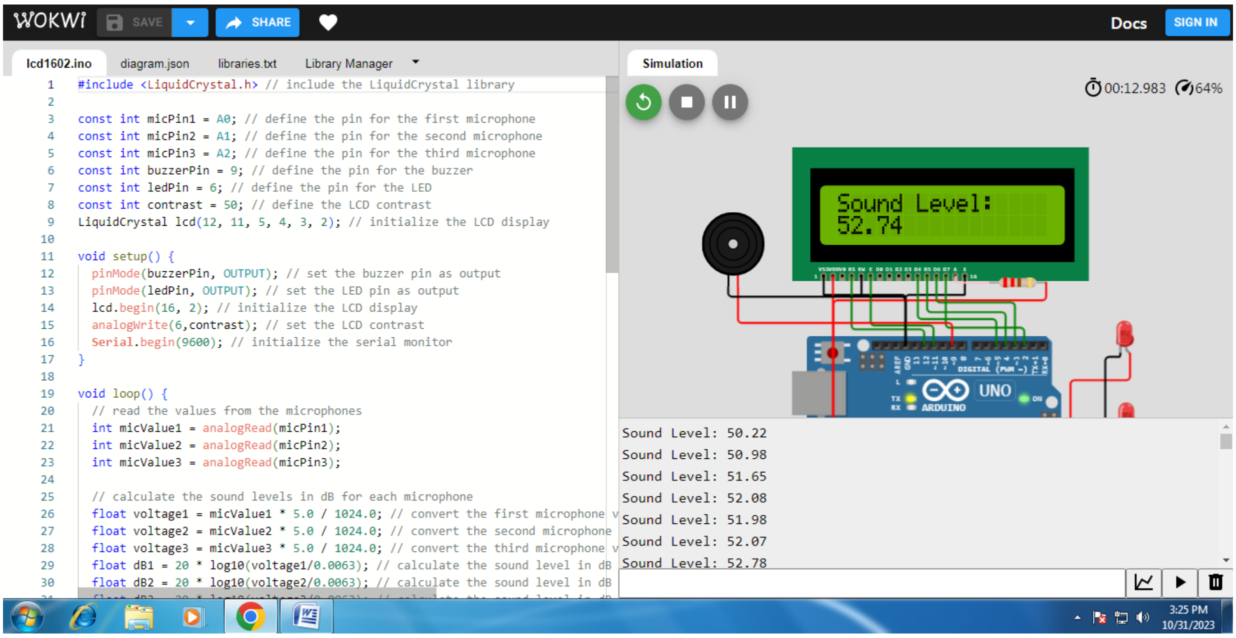
```
<!DOCTYPE html>
<html>
<head>
    <title>AQI Data from Firebase</title>
    <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-database.js"></script>
</head>
<body>
    <h1>Air Quality Index (AQI) Data</h1>
    <div id="aqi-data">
        <!-- AQI data will be displayed here -->
    </div>

    <script>
        // Initialize Firebase with your project's
        configuration var firebaseConfig = {
            apiKey: "YOUR_API_KEY",
            authDomain:
            "YOUR_AUTH_DOMAIN",
            databaseURL: "YOUR_DATABASE_URL",
            projectId: "YOUR_PROJECT_ID",
            storageBucket:
            "YOUR_STORAGE_BUCKET",
            messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
            appId: "YOUR_APP_ID"
        };

        firebase.initializeApp(firebaseConfig);
```

```
      // Reference to your AQI data in
      Firebase var aqiRef =
      firebase.database().ref("aqi");

      // Listen for changes in the AQI
      data aqiRef.on("value",
      function(snapshot) {
        var aqiData = snapshot.val();

        // Update the HTML to display the AQI
        data if (aqiData) {
          document.getElementById("aqi-data").innerHTML = "AQI: " + aqiData;
        } else {
          document.getElementById("aqi-data").innerHTML = "No data available";
        }
      });
    </script>
  </body>
  </html>
```

## NOISE POLLUTION MONITORING CODE:

```
#include <LiquidCrystal.h> // include the LiquidCrystal library

const int micPin1 = A0; // define the pin for the first microphone
const int micPin2 = A1; // define the pin for the second microphone
const int micPin3 = A2; // define the pin for the third microphone
const int buzzerPin = 9; // define the pin for the buzzer
const int ledPin = 6; // define the pin for the LED
const int contrast = 50; // define the LCD contrast
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // initialize the LCD display

void setup() {
  pinMode(buzzerPin, OUTPUT); // set the buzzer pin as output
  pinMode(ledPin, OUTPUT); // set the LED pin as output
  lcd.begin(16, 2); // initialize the LCD display
  analogWrite(6,contrast); // set the LCD contrast
  Serial.begin(9600); // initialize the serial monitor
}

void loop() {
  // read the values from the microphones
  int micValue1 = analogRead(micPin1);
  int micValue2 = analogRead(micPin2);
  int micValue3 = analogRead(micPin3);

  // calculate the sound levels in dB for each microphone
  float voltage1 = micValue1 * 5.0 / 1024.0; // convert the first microphone value to voltage (5V reference)
  float voltage2 = micValue2 * 5.0 / 1024.0; // convert the second microphone value to voltage (5V reference)
  float voltage3 = micValue3 * 5.0 / 1024.0; // convert the third microphone value to voltage (5V reference)
  float dB1 = 20 * log10(voltage1/0.0063); // calculate the sound level in dB for the first microphone
  float dB2 = 20 * log10(voltage2/0.0063); // calculate the sound level in dB for the second microphone
  float dB3 = 20 * log10(voltage3/0.0063); // calculate the sound level in dB for the third microphone

  // calculate the average sound level in dB for all microphones
  float averageDB = (dB1 + dB2 + dB3) / 3;

  // display the sound level on the LCD display and the serial monitor
  lcd.setCursor(0, 0); // set the cursor to the first row of the LCD display
  lcd.print("Sound Level: "); // print the text "Sound Level: " on the LCD display
  lcd.setCursor(0, 1); // set the cursor to the second row of the LCD display
  lcd.print(averageDB); // print the average sound level on the LCD display
  Serial.print("Sound Level: "); // print the text "Sound Level: " on the serial monitor
  Serial.println(averageDB); // print the average sound level on the serial monitor

  // control the LED and the buzzer based on the sound level
  if (averageDB > 70) { // if the sound level is higher than 70 dB
    digitalWrite(ledPin, HIGH); // turn the LED on
    tone(buzzerPin, 1000, 500); // turn the buzzer on
  } else { // if the sound level is lower than 70 dB
    digitalWrite;
  }
}
```

## RESULT:



```
#include <LiquidCrystal.h> // include the LiquidCrystal library

const int micPin1 = A0; // define the pin for the first microphone
const int micPin2 = A1; // define the pin for the second microphone
const int micPin3 = A2; // define the pin for the third microphone
const int buzzerPin = 9; // define the pin for the buzzer
const int ledPin = 6; // define the pin for the LED
const int contrast = 50; // define the LCD contrast
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // initialize the LCD display

void setup() {
  pinMode(buzzerPin, OUTPUT); // set the buzzer pin as output
  pinMode(ledPin, OUTPUT); // set the LED pin as output
  lcd.begin(16, 2); // initialize the LCD display
  analogWrite(6,contrast); // set the LCD contrast
  Serial.begin(9600); // initialize the serial monitor
}

void loop() {
  // read the values from the microphones
  int micValue1 = analogRead(micPin1);
  int micValue2 = analogRead(micPin2);
  int micValue3 = analogRead(micPin3);

  // calculate the sound levels in dB for each microphone
  float voltage1 = micValue1 * 5.0 / 1024.0; // convert the first microphone v
  float voltage2 = micValue2 * 5.0 / 1024.0; // convert the second microphone
  float voltage3 = micValue3 * 5.0 / 1024.0; // convert the third microphone v
  float dB1 = 20 * log10(voltage1/0.0063); // calculate the sound level in dB
  float dB2 = 20 * log10(voltage2/0.0063); // calculate the sound level in dB
  float dB3 = 20 * log10(voltage3/0.0063); // calculate the sound level in dB
```

```
Sound Level: 50.22
Sound Level: 50.98
Sound Level: 51.65
Sound Level: 52.08
Sound Level: 51.98
Sound Level: 52.07
Sound Level: 52.78
```

## CONCLUSION:

In conclusion, noise pollution monitoring is a critical aspect of urban and environmental management. It helps us understand and mitigate the adverse effects of excessive noise on human health and the environment.