1. Main code

#define printInterval 800

float pHValue;

void setup() {

  setupDisplay();

  setupTemp();

  setupLevel();

}

void loop() {

  float level = readLevel();

  float temp = readTemp();

  readPH();

  static unsigned long printTimepoint = millis();

  if (millis() - printTimepoint > printInterval) {

    printTimepoint = millis();

    print(level, temp, pHValue);

  }

}

1. LCD Display

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

/// Define the I2C address of the LCD

const int LCD\_ADDR = 0x27;

/// Initialize the LCD with I2C interface

LiquidCrystal\_I2C lcd(LCD\_ADDR, 20, 4);

/// Prints a text in the first line of the LCD

void printA(const char \*text) {

  lcd.setCursor(0, 0);

  lcd.print(text);

}

/// Prints a text in the first line of the LCD in a given position

void printA(const char \*text, uint8\_t pos) {

  lcd.setCursor(pos, 0);

  lcd.print(text);

}

/// Prints a text in the second line of the LCD

void printB(const char \*text) {

  lcd.setCursor(0, 1);

  lcd.print(text);

}

/// Prints a text in the second line of the LCD in a given position

void printB(const char \*text, uint8\_t pos) {

  lcd.setCursor(pos, 1);

  lcd.print(text);

}

/// Prints a text in the third line of the LCD

void printC(const char \*text) {

  lcd.setCursor(0, 2);

  lcd.print(text);

}

/// Prints a text in the third line of the LCD in a given position

void printC(const char \*text, uint8\_t pos) {

  lcd.setCursor(pos, 2);

  lcd.print(text);

}

/// Prints a text in the fourth line of the LCD

void printD(const char \*text) {

  lcd.setCursor(0, 3);

  lcd.print(text);

}

/// Prints a text in the fourth line of the LCD in a given position

void printD(const char \*text, uint8\_t pos) {

  lcd.setCursor(pos, 3);

  lcd.print(text);

}

/// Prints the values after reading from the sensors

void print(float level, float temp, float pH) {

  char lbuf[10];

  char tbuf[10];

  char pbuf[10];

  char \*l = dtostrf(level, 4, 2, lbuf);

  strcat(lbuf, "cm");

  char \*t = dtostrf(temp, 4, 2, tbuf);

  strcat(tbuf, "C");

  char \*p = dtostrf(pH, 4, 2, pbuf);

  printA("       ", 13);

  printA(l, 13);

  printB("           ", 6);

  printB(t, 6);

  printC("          ", 10);

  printC(p, 10);

}

/// Setup the LCD display

void setupDisplay() {

  // Initialize the LCD

  lcd.init();

  lcd.backlight();

  printA(" Arduino Uno Water");

  printB(" Quality Monitoring");

  printC("System Temperature &");

  printD("    pH Detector     ");

  delay(5000);

  lcd.clear();

  printA("Water level: ");

  printB("Temp: ");

  printC("pH value: ");

}

1. Water Level Indicator

/// Height of the container in cm

const int CONTAINER\_HEIGHT = 25;

/// Define the pins for the HC-SR04 ultrasonic sensor

const int TRIG\_PIN = 3;

const int ECHO\_PIN = 4;

/// Setup the Level indicator (Ultrasonic Sensor)

void setupLevel() {

  // Initialize the HC-SR04 ultrasonic sensor

  pinMode(TRIG\_PIN, OUTPUT);

  pinMode(ECHO\_PIN, INPUT);

}

/// Read the water level

float readLevel() {

  // Read the distance value from the HC-SR04 sensor

  digitalWrite(TRIG\_PIN, LOW);

  delayMicroseconds(2);

  digitalWrite(TRIG\_PIN, HIGH);

  delayMicroseconds(10);

  digitalWrite(TRIG\_PIN, LOW);

  float duration = pulseIn(ECHO\_PIN, HIGH);

  float distance = duration \* 0.034 / 2;

  float level = CONTAINER\_HEIGHT - distance;

  return level > 0 ? level : 0;

}

1. PH Sensor

#define SensorPin A0  //pH meter Analog output to Arduino Analog Input 0

#define Offset -0.35  //deviation compensate

#define LED 13

#define samplingInterval 20

#define ArrayLenth 40       //times of collection

int pHArray[ArrayLenth];  //Store the average value of the sensor feedback

int pHArrayIndex = 0;

void readPH(void) {

  static unsigned long samplingTime = millis();

  static float voltage;

  if (millis() - samplingTime > samplingInterval) {

    pHArray[pHArrayIndex++] = analogRead(SensorPin);

    if (pHArrayIndex == ArrayLenth) pHArrayIndex = 0;

    voltage = avergearray(pHArray, ArrayLenth) \* 5.0 / 1024;

    pHValue = 3.5 \* voltage + Offset;

    samplingTime = millis();

  }

}

double avergearray(int\* arr, int number) {

  int i;

  int max, min;

  double avg;

  long amount = 0;

  if (number <= 0) {

    Serial.println("Error number for the array to avraging!/n");

    return 0;

  }

  if (number < 5) {  //less than 5, calculated directly statistics

    for (i = 0; i < number; i++) {

      amount += arr[i];

    }

    avg = amount / number;

    return avg;

  } else {

    if (arr[0] < arr[1]) {

      min = arr[0];

      max = arr[1];

    } else {

      min = arr[1];

      max = arr[0];

    }

    for (i = 2; i < number; i++) {

      if (arr[i] < min) {

        amount += min;  //arr<min

        min = arr[i];

      } else {

        if (arr[i] > max) {

          amount += max;  //arr>max

          max = arr[i];

        } else {

          amount += arr[i];  //min<=arr<=max

        }

      }  //if

    }    //for

    avg = (double)amount / (number - 2);

  }  //if

  return avg;

}

1. Temperature Sensor

#include <OneWire.h>

#include <DallasTemperature.h>

/// Define the pin for the DS18B20 temperature sensor

const int TEMP\_PIN = 2;

/// Initialize the one-wire interface for the temperature sensor

OneWire oneWire(TEMP\_PIN);

/// Initialize the DallasTemperature library

DallasTemperature sensors(&oneWire);

/// Setup Temperature sensor (DS18B21)

void setupTemp() {

  // Initialize the temperature sensor

  sensors.begin();

}

/// Read temperature value

float readTemp() {

  // Read the temperature value from the DS18B20 sensor

  sensors.requestTemperatures();

  float temperature = sensors.getTempCByIndex(0);

  return temperature;

}