Indentation:

```
#include <SoftwareSerial.h>

#include <LiquidCrystal.h>

                                        //(Remove unnecessary comments)

SoftwareSerial sserial(12, 13);

LiquidCrystal lcd(9, 8, 7, 6, 5, 4);


#define turbidityPin A2

#define valvePin 11

#define flowPin 2

 //(Remove unnecessary lines)

float turbidity = 0;

float flowRate = 0;

bool leak = 0;

bool valve = 0;

int waterLimit = 10;

int waterUsed = 0;


volatile byte pulseCount;   (Remove unnecessary lines)


void pulseCounter() {     //(Remove unnecessary line and combine "{" to previous "( )"  )

  pulseCount++;

}

 //(Remove unnecessary lines)

void setup() {      //(Remove unnecessary line and combine "{" to previous "( )"  )

  Serial.begin(115200);

  sserial.begin(9600);     //(Remove unnecessary line)

  pinMode(valvePin, OUTPUT);
  digitalWrite(valvePin, LOW);    //(Remove unnecessary line)
  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("  Water Meter   ");
  delay(5000);
  lcd.clear();
  digitalWrite(valvePin, HIGH);    // (Remove unnecessary line)
  pinMode(flowPin, INPUT_PULLUP);
  attachInterrupt(0, pulseCounter, FALLING);
}
//(Remove unnecessary lines)
void loop() {     (Remove unnecessary line and combine “{” to previous “( )”  )
  measureTurbidity();
  updateLCD();
  measureFlow();
  detectLeakage();
  sendData();  //(Remove delay(10) isn’t necessary)
}
 //(Remove unnecessary line and combine “{” to previous “( )”  )
void detectLeakage() {
  uint32_t interval_ms = 60000;
  static uint32_t time3;  //(Remove unnecessary line)
  if (millis() - time3 > interval_ms) {   //(Combine “{” to previous “( )”  )
    digitalWrite(valvePin, HIGH);
    if (millis() - time3 > interval_ms + 5000) {  //(Combine “{” to previous “( )”  )
      bool leak_ = (flowRate > 0.00f);
      if (millis() - time3 > interval_ms + 5000) {  //(Combine “{” to previous “( )”  )
        leak = leak_;
        time3 = millis();
      }
    }
  } else {
    valve = (waterUsed >= waterLimit || turbidity >= 3000 || leak) ? 0 : 1;
    digitalWrite(valvePin, valve ? LOW : HIGH);
  }
}
 //(Remove unnecessary line)
void measureFlow() {  //(Remove unnecessary line and combine “{” to previous “( )”  )
  static uint32_t previousMillis;
  static uint32_t totalMilliLitres;
  static float totalLitres ;
  if (millis() - previousMillis > 1000) {   //(Remove unnecessary line and combine “{” to
previous “( )”  )
    uint16_t pulse1Sec = pulseCount;
    pulseCount = 0;   //(Remove unnecessary line)
    flowRate = ((1000.0 / (millis() - previousMillis)) * pulse1Sec) / 4.5;
    previousMillis = millis();
    uint32_t flowMilliLitres = (flowRate / 60) * 1000;
    float flowLitres = (flowRate / 60);
    totalMilliLitres += flowMilliLitres;
    totalLitres += flowLitres;
    waterUsed = totalLitres;  //(Remove unnecessary line )
    Serial.println(flowRate);
    Serial.println(waterUsed);
  }
}
 //(Remove unnecessary line and combine “{” to previous “( )”  )
void measureTurbidity() {
  float volt = 0;
  for (int i = 0; i < 100; i++) {
    volt += (analogRead(turbidityPin) / 1023.0) * 5;
    delay(5);
  }
  volt /= 100;
  volt = round_to_dp(volt, 2);
  if (volt < 2.5) turbidity = 3000;
  else turbidity = -1120.4 * sq(volt) + 5742.3 * volt - 4353.8;
}   //(Remove unnecessary comments)
 //(Remove unnecessary line and combine “{” to previous “( )”  )
void updateLCD() {
  static uint32_t time1;
  static byte screen = 0;   //(Remove unnecessary line)
  if (millis() - time1 > 3000) {
    lcd.clear();
    time1 = millis();
    screen++;
    if (screen > 2) screen = 0;
  }  //(Remove unnecessary line)
  switch (screen) {
    case 0:
      lcd.setCursor(0, 0);
      lcd.print("   Turbidity    ");
      lcd.setCursor(0, 1);
      lcd.print("  ");
      lcd.print(turbidity);
      lcd.print(" NTU");
      break;
    case 1:
      lcd.setCursor(0, 0);
      lcd.print(leak ? "Leakage Detected" : "   No Leakage   ");
      lcd.setCursor(0, 1);
      lcd.print(valve ? "  Valve Opened  " : "  Valve Closed  ");
      break;
    case 2:
      lcd.setCursor(0, 0);
      lcd.print("H2O limit: ");
      lcd.print(waterLimit);
      lcd.print("L ");
      lcd.setCursor(0, 1);
      lcd.print("H2O Used: ");
      lcd.print(waterUsed);
      lcd.print("L ");
      break;
  }
}   //(Remove unnecessary line and combine “{” to previous “( )”  )

void sendData() {
  static uint32_t time1;
  if (millis() - time1 > 1000) {  //(Combine “{” to previous “( )”  )
    sserial.print(
      "{\"t\":" + (String)turbidity +
      ",\"l\":" + (String)leak +
      ",\"v\":" + (String)valve +
      ",\"wl\":" + (String)waterLimit +
      ",\"wu\":" + (String)waterUsed +
      "}");
    time1 = millis();
  }
}    //(Remove unnecessary line and combine “{” to previous “( )”  )
```

```
float round_to_dp(float in_value, int decimal_place) {  //(Combine “{” to previous “( )”  )
  float multiplier = powf(10.0f, decimal_place);
  in_value = roundf(in_value * multiplier) / multiplier;
  return in_value;
}
```

Conciseness:

```
#include <SoftwareSerial.h>
SoftwareSerial sserial(12, 13);

#include <LiquidCrystal.h>
LiquidCrystal lcd(9, 8, 7, 6, 5, 4);

#define TURBIDITY_PIN A2
#define VALVE_PIN 11
#define FLOW_PIN 2

float turbidity = 0, flowRate = 0;
bool leak = 0, valve = 0;
int waterLimit = 10, waterUsed = 0;

void setup() {
  Serial.begin(115200);
  sserial.begin(9600);
  pinMode(VALVE_PIN, OUTPUT);   // Removed digitalWrite(valvePin, LOW);
  lcd.begin(16, 2);             // Removed lcd.print("  Water Meter   ");
  attachInterrupt(0, pulseCounter, FALLING);   // Removed delay(5000);  // Removed
lcd.clear();
}                               // Removed digitalWrite(valvePin, HIGH);
                               // pinMode(flowPin, INPUT_PULLUP);
```

```
void loop() {
  Turbidity();
  lcdScr();
  flow();
  leakage();
  sendData();   // Removed delay(10);
}

void leakage() {
  const uint32_t intervalMs = 60000;
  static uint32_t time3;
  if (millis() - time3 > intervalMs) {    // Removed digitalWrite{valvePin, 1}
    leak = (flowRate > 0.00f);            // Removed leak = leak_;
    valve = (waterUsed >= waterLimit || turbidity >= 3000 || leak);
    digitalWrite(VALVE_PIN, valve ? LOW : HIGH);   // Removed time3 = millis();
    previousTime = millis();
  }
}

void pulseCounter() {
  static byte pulseCount;
  pulseCount++;
  flowRate = ((1000.0 / (millis() - previousMillis)) * pulseCount) / 4.5;
  waterUsed += (flowRate / 60) * 1000;
  pulseCount = 0;    // Flow function is also used in this function
}

void Turbidity() {
  float volt = 0;
  for (int i = 0; i < 100; i++) {
    volt += ((float)analogRead(TURBIDITY_PIN) / 1023) * 5;
    delay(5);
  }
  volt /= 100;
  turbidity = (volt < 2.5) ? 3000 : -1120.4 * sq(volt) + 5742.3 * volt - 4353.8;
}          // Removed unnecessary comments

void lcdScr() {
  // Removed static uint32_t time1;
  // if (millis() - time1 > 3000) {
  //  lcd.clear();
  //  time1 = millis();
  //  screen = (screen + 1) % 3;
  //  }

  static byte screen;
  lcd.clear();
  switch (screen) {
    case 0:                  // Removed lcd.setCursor(0,0)
      lcd.print("Turbidity: ");  // Removed lcd.setCursor(0,1)
      lcd.print(turbidity);     // Removed lcd.print(" ")
      lcd.println(" NTU");       // NPU stands for Nephelometric Turbidity Units
      break;     // Removed unnecessaey line
    case 1:            // Removed lcd.setCursor(0,0)
      lcd.print(leak ? "Leakage Detected" : "No Leakage");  // Removed lcd.setCursor(0,1)
      lcd.println();
      lcd.print(valve ? "Valve Opened" : "Valve Closed");
      break;
    case 2:
      lcd.print("H2O limit: ");   // Removed lcd.setCursor(0,0)
    lcd.print(waterLimit);
    lcd.println("L");         // Removed lcd.setCursor(0,1)
    lcd.print("H2O Used: ");
    lcd.print(waterUsed);
    lcd.println("L");
    break;
  }
  screen = (screen + 1) % 3;
}

void sendData() {
  static uint32_t time1;
  if (millis() - previousTime > 1000) {
    sserial.print("{\"t\":" + (String)turbidity + ",\"l\":" + (String)leak + ",\"v\":" + (String)valve + ",\"wl\":" + (String)waterLimit + ",\"wu\":" + (String)waterUsed + "}");
    time1 = millis();    // Removed spaces and merge above attributes
  }
}
```

NAMING FUNCTION:

Functions:	Names:
1. `pulseCounter()`	`countPulses()`
2. `setup()`	`initializeSystem()`
3. `loop()`	`runSystem()`
4. `Leakage()`	`checkForLeaks()`
5. `Flow()`	`calculateFlowRate()`
6. `Turbidity()`	`calculateTurbidity()`
7. `LCDScr()`	`displayDataOnLCD()`
8. `sendData()`	`transmitDataToSerial()`

9. `round_to_dp()`   `roundToDecimalPlaces()`

Syntax:

The syntax of the provided code is written in C++. Here's a breakdown of the syntax elements used:

1. **Preprocessor Directives**:

   - `#include <SoftwareSerial.h>`: Includes the header file for SoftwareSerial library.

   - `#include <LiquidCrystal.h>`: Includes the header file for LiquidCrystal library.

   - `#define`: Defines constants for pin numbers (`turbidityPin`, `valvePin`, `flowPin`).

2. **Variable Declaration**:

   - `float`, `bool`, and `int` data types are used to declare variables (`turbidity`, `flowRate`, `leak`, `valve`, `waterLimit`, `waterUsed`).

   - `volatile byte pulseCount`: Declares a volatile variable to count pulses.

3. **Function Declarations**:

   - Functions like `pulseCounter()`, `setup()`, `loop()`, `detectLeakage()`, `measureFlow()`, `measureTurbidity()`, `updateLCD()`, `sendData()`, and `round_to_dp()` are declared.

4. **Function Definitions**:

   - Function definitions provide the implementation for each declared function.

5. **Control Structures**:

   - `if`, `else`, and `switch` statements are used for control flow.

   - `for` loop is used in the `measureTurbidity()` function.

6. **Pin Modes**:

   - `pinMode()` function sets the pin mode for `valvePin` and `flowPin`.

7. **I/O Functions**:

- Functions like `digitalWrite()` and `analogRead()` are used to interact with digital and analog pins respectively.

8. **Library Functions**:

- Functions like `begin()` and `clear()` are used to initialize and clear the LCD display.

9. **Serial Communication**:

- `Serial.begin()` and `sserial.begin()` functions initialize serial communication.
- `sserial.print()` is used to transmit data via SoftwareSerial.

10. **Mathematical Functions**:

- Functions like `roundf()` and `powf()` are used for mathematical calculations.

11. **Data Types**:

- The code uses data types like `uint32_t`, `uint16_t`, `byte`, `float`, `bool`, `int`, and `String` to declare variables and define function parameters.