ECET 230 - Test Codes

Rachel Romeo, Hailey Horvath

2025-12-20

Table of Contents

What are Test Codes

Before starting a project, it is important to verify that all components that are supposed to be used work. Test codes are short, simple programs written to verify that each individual component in a system is working correctly before integrating everything into the full project. They allow for checking wiring, quickly power connections, and basic functionality, such as whether a sensor outputs changing values or an LED lights up. By running test codes, it can isolate problems early and avoid spending hours debugging a large program when the issue might just be a loose wire or faulty component. In other words, test codes make troubleshooting easier and ensure all parts are reliable before combining them into the final design.

Proof-of-Concept Testing

For our proof of concept, we focused on testing whether each sensor and the RGB LED could correctly respond to environmental changes. We did not use a real plant during this stage. Instead, we used simple methods like touching the soil moisture sensor with our hands, breathing near the temperature sensor, and covering or uncovering the photoresistor to simulate different conditions.

- The capacitive soil moisture sensor reacted as expected when we touched it. The readings increased when our hands made contact, showing that the sensor was detecting a change in moisture levels. This confirmed that the sensor's analog input and mapping function in the code were working properly.
- For the DHT11 temperature sensor, we tested it by breathing near it to raise the temperature slightly. The Serial Monitor showed the temperature increasing as we did this, confirming that the sensor could detect small temperature changes accurately and communicate them to the Arduino.
- The photoresistor was tested by covering and uncovering it to simulate changes in light. The readings decreased when the sensor was covered and increased under brighter light, showing that it was responding properly to variations in brightness.
- The RGB LED indicator tested the different colors, which were programmed to represent specific alerts. When we simulated various sensor conditions, the LED changed colors as expected. Red for high temperature, blue for low temperature, yellow for dry soil, and purple for low light. This confirmed that the LED output logic and color control worked correctly.

Overall, these tests showed that all sensors and the RGB LED functioned as intended. The system successfully detected and displayed changes in the

environment, confirming that our code and wiring were working before moving on to testing with an actual plant.

The following summarizes the tests conducted prior to the Proof-of-Concept Prototype Midterm Presentation:

Test and Results Table:

Test	Method	Expected Outcome	Actual Result	Pass Fail
Moisture	Touching the sensor with a hand	Higher Moisture Reading due to moisture on the hand	Increased Reading	Pass
Temperature	Breathe on Sensor	Temperature increase due to hot breath	Increased Reading	Pass
Light	Cover Sensor	Light level decrease due to less light exposure	Decreased Reading	Pass
RGB LED	Simulate Alert	Correct Color lights up based on the current measurements and optimal ranges inputted	Color matched the current condition of the plant (Ex. Too cold = Blue LED)	Pass

Test Codes

Arduino Mega Test (Basic Board Test)

```
// Blinks the built-in LED to confirm the Arduino Mega works
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // LED on
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);  // LED off
  delay(500);
}
```

DHT11 Temperature & Humidity Sensor Test

```
#include "DHT.h"

#define DHTPIN 2       // Digital pin connected to DHT11
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();

  if (isnan(temp) || isnan(hum)) {
    Serial.println("Failed to read from DHT11 sensor!");
    return;
  }

  Serial.print("Temp: ");
  Serial.print(temp);
```

```
  Serial.print(" °C  |  Humidity: ");
  Serial.print(hum);
  Serial.println(" %");
  delay(2000);
}
```

Capacitive Soil Moisture Sensor Test

```
const int soilSensorPin = A0; // Connect sensor's analog output

void setup() {
  Serial.begin(9600);
  pinMode(soilSensorPin, INPUT);
  Serial.println("Capacitive Soil Moisture Sensor Test");
}

void loop() {
  int sensorValue = analogRead(soilSensorPin);
  int moisturePercent = map(sensorValue, 1023, 0, 0, 100); // 0% dry, 100% wet
  Serial.print("Analog Value: ");
  Serial.print(sensorValue);
  Serial.print("\tMoisture: ");
  Serial.print(moisturePercent);
  Serial.println("%");
  delay(500);
}
```

Photoresistor (LDR) Test:

```
#define LDR_PIN A1

void setup() {
  Serial.begin(9600);
}

void loop() {
  int lightValue = analogRead(LDR_PIN);
  Serial.print("Light Level: ");
  Serial.println(lightValue);
  delay(1000);
}
```

RGB LED Test:

```
#define RED_PIN 9
#define GREEN_PIN 10
#define BLUE_PIN 11

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop() {
  // Red
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 0);
  delay(1000);

  // Green
  analogWrite(RED_PIN, 0);
  analogWrite(GREEN_PIN, 255);
  analogWrite(BLUE_PIN, 0);
  delay(1000);

  // Blue
  analogWrite(RED_PIN, 0);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 255);
  delay(1000);

  // White
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 255);
  analogWrite(BLUE_PIN, 255);
  delay(1000);
}
```

16×2 LCD + Potentiometer Test

```
#include <LiquidCrystal.h>

// RS, E, D4, D5, D6, D7
```

```
LiquidCrystal lcd(30, 31, 32, 33, 34, 35);

void setup() {
  lcd.begin(16, 2);
  lcd.print("LCD Test");
}

void loop() {
  lcd.setCursor(0, 1);
  lcd.print(millis() / 1000); // Display elapsed seconds
  delay(1000);
}
```

4×4 Keypad Test

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {22, 23, 24, 25};
byte colPins[COLS] = {26, 27, 28, 29};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

void setup() {
  Serial.begin(9600);
  Serial.println("Keypad Test: Press keys");
}

void loop() {
  char key = keypad.getKey();
  if (key) {
    Serial.print("Key Pressed: ");
    Serial.println(key);
  }
}
```

Reset Button Test

```
#define RESET_PIN 7

void setup() {
  pinMode(RESET_PIN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  if (digitalRead(RESET_PIN) == LOW) {
    Serial.println("Reset button pressed!");
    delay(500); // debounce
  }
}
```

Proof-of-Concept Prototype Midterm Presentation:

```
#include "DHT.h"

// Pins
const int soilSensorPin = A0;     // Capacitive Soil Moisture Sensor
const int lightSensorPin = A1;    // Photoresistor
const int DHTPin = 2;             // DHT11 data pin
const int redPin = 9;             // RGB LED Red
const int greenPin = 10;          // RGB LED Green
const int bluePin = 11;           // RGB LED Blue

#define DHTTYPE DHT11
DHT dht(DHTPin, DHTTYPE);

// Sensor ranges
int soilLower = 0;      // Only a lower limit now
float tempLower = 0;
float tempUpper = 0;
int lightThreshold = 0;

// Nighttime floor for light sensor
const int nightLevel = 10; // anything below this is considered night

// Timing
unsigned long previousMillis = 0;
const unsigned long interval = 10000; // 10 seconds

void setup() {
  Serial.begin(9600);
  pinMode(soilSensorPin, INPUT);
  pinMode(lightSensorPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
```

```
  pinMode(bluePin, OUTPUT);
  dht.begin();

  Serial.println("Smart Plant Monitor with RGB LED Alerts (°F)");

  // --- Soil Moisture Input ---
  while (true) {
    Serial.println("Enter the minimum (LOWER) soil moisture level
(0-100):");
    while (Serial.available() == 0) {}
    soilLower = Serial.parseInt();
    while (Serial.available() > 0) { Serial.read(); }
    if (soilLower >= 0 && soilLower <= 100) break;
    Serial.println("Invalid input. Enter 0-100.");
  }

  // --- Temperature Input (0-100°F) ---
  while (true) {
    Serial.println("Enter the LOWER optimal temperature (°F, 0-100):");
    while (Serial.available() == 0) {}
    tempLower = Serial.parseFloat();
    while (Serial.available() > 0) { Serial.read(); }
    if (tempLower >= 0 && tempLower <= 100) break;
    Serial.println("Invalid input. Enter a temperature between 0 and 100
°F.");
  }

  while (true) {
    Serial.println("Enter the UPPER optimal temperature (°F, 0-100):");
    while (Serial.available() == 0) {}
    tempUpper = Serial.parseFloat();
    while (Serial.available() > 0) { Serial.read(); }
    if (tempUpper >= tempLower && tempUpper <= 100) break;
    Serial.println("Invalid input. Must be >= lower and <= 100 °F.");
  }

  // --- Light Threshold Input ---
```

```
  while (true) {
    Serial.println("Enter the minimum optimal light level (0-1023):");
    while (Serial.available() == 0) {}
    lightThreshold = Serial.parseInt();
    while (Serial.available() > 0) { Serial.read(); }
    if (lightThreshold >= 0 && lightThreshold <= 1023) break;
    Serial.println("Invalid input. Enter a value between 0 and 1023.");
  }

  Serial.println("---- START MONITORING EVERY 10 SECONDS ----");
}

// Helper function to set RGB LED
void setLED(float r, float g, float b) {
  analogWrite(redPin, (int)(r * 255));
  analogWrite(greenPin, (int)(g * 255));
  analogWrite(bluePin, (int)(b * 255));
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    // --- Read sensors ---
    int soilValue = analogRead(soilSensorPin);
    int soilPercent = map(soilValue, 1023, 0, 0, 100);

    float temperatureC = dht.readTemperature();
    float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;

    int lightValue = analogRead(lightSensorPin);

    // --- Print readings ---
    Serial.print("Soil Moisture: "); Serial.print(soilPercent);
Serial.print("%");
```

```
    if (soilPercent < soilLower) Serial.print(" --> TOO DRY!");
    Serial.println();

    Serial.print("Temperature: "); Serial.print(temperatureF);
Serial.print(" °F");
    if (temperatureF < tempLower || temperatureF > tempUpper)
Serial.print(" --> OUT OF OPTIMAL TEMPERATURE RANGE!");
    Serial.println();

    Serial.print("Light Level: "); Serial.print(lightValue);
    // Only alert for low light if above nightLevel
    if (lightValue < lightThreshold && lightValue > nightLevel)
Serial.print(" --> TOO DARK!");
    Serial.println();

    Serial.println("---------------------------------------");

    // --- Determine issues ---
    bool highTemp = (temperatureF > tempUpper);
    bool lowTemp  = (temperatureF < tempLower);
    bool drySoil  = (soilPercent < soilLower);
    bool lowLightFlag = (lightValue < lightThreshold && lightValue >
nightLevel); // Night-aware

    // --- LED Logic ---
    int activeIssues = highTemp + lowTemp + drySoil + lowLightFlag;

    if (activeIssues == 0) {
        setLED(0,0,0); // Off
    }
    else if (activeIssues == 1) {
        if (highTemp) setLED(1,0,0);         // Red
        else if (lowTemp) setLED(0,0,1);     // Blue
        else if (drySoil) setLED(1,1,0);     // Yellow
        else if (lowLightFlag) setLED(1,0,1); // Purple
    }
    else {
```

```
        // Multiple issues: blink all active colors up to 6 cycles
        for (int blinkCount = 0; blinkCount < 6; blinkCount++) {
            if (highTemp) { setLED(1,0,0); delay(800); setLED(0,0,0);
delay(400); }
            if (lowTemp)  { setLED(0,0,1); delay(800); setLED(0,0,0);
delay(400); }
            if (drySoil)  { setLED(1,1,0); delay(800); setLED(0,0,0);
delay(400); }
            if (lowLightFlag) { setLED(1,0,1); delay(800); setLED(0,0,0);
delay(400); }
        }
        // After blinking, set LED to reflect current state
        if (!highTemp && !lowTemp && !drySoil && !lowLightFlag)
setLED(0,0,0);
        else if (highTemp) setLED(1,0,0);
        else if (lowTemp)  setLED(0,0,1);
        else if (drySoil)  setLED(1,1,0);
        else if (lowLightFlag) setLED(1,0,1); // Purple
    }
  }
}
```

Final Project Code:

Final Project Code Summary

This Arduino code is the main program for a Smart Plant Monitor. It reads data from multiple sensors (soil moisture, temperature & humidity, and light), allows the user to configure optimal ranges via a keypad, displays status on a 16×2 LCD, and uses an RGB LED to provide visual feedback about plant conditions. A reset button allows the user to restart the system.

The code supports two modes:

- REAL_MODE: Uses normal time intervals (minutes/hours) for plant monitoring.
- TEST_MODE: Uses shorter time intervals (seconds) for testing and debugging.

To switch between these two modes, the code must be slightly changed. In roughly lines 5-6, there is:

- *//#define TEST_MODE*
- *#define REAL_MODE*

To toggle between these two modes, comment out the def of the mode that is not wanted.

Libraries

- DHT.h → For reading DHT11 temperature & humidity sensor.
- Keypad.h → For handling input from a 4×4 keypad.
- LiquidCrystal.h → For controlling a 16×2 LCD.

Key Functional Sections

- Keypad Input
    - Allows the user to enter numeric values for:
    - Minimum soil moisture (soilLower)
    - Optimal temperature range (tempLower and tempUpper)
    - Minimum light level (lightThreshold)
    - Daily sunlight goal (sunlightGoalSeconds)
    - Monitoring interval (userInterval)
    - Supports:
        - A → Enter / confirm
        - B → Backspace
        - C → Clear / restart inputs
- LCD Output
    - Displays prompts and input values when configuring the system.
    - Shows temporary confirmation messages after input.
- Reset Button Handling

    - Detects long press (5 seconds) to restart the system.
    - Stops the RGB LED and re-initializes all variables by calling setup().
- Sensor Reading
    - Soil Moisture → Read analog value, map to percentage (0–100%).
    - Temperature → DHT11 in °F.
    - Light → Analog LDR reading, used to track sunlight exposure.
- Sunlight Tracking
    - Tracks accumulated sunlight time for the day.
    - Resets sunlight counter if prolonged darkness is detected (nightResetSeconds).
    - Compares accumulated sunlight to user-defined goal (sunlightGoalSeconds).
- RGB LED Feedback
    - Displays plant status visually using colors:
        - Red → High temperature
        - Blue → Low temperature
        - Yellow → Soil too dry
        - Purple → Light below threshold
    - Handles multiple active issues by blinking each corresponding color in sequence.
    - Uses non-blocking timing with millis() for smooth LED blinking
- Intervals
    - The interval controls how often the system reads sensors and updates statuses.
    - Can be set in seconds (TEST_MODE) or minutes (REAL_MODE).
    - Non-blocking loops allow simultaneous LED blinking and sensor monitoring.

Below is a simplified code flow chart, which highlights how the system will behave if certain requirements are met:

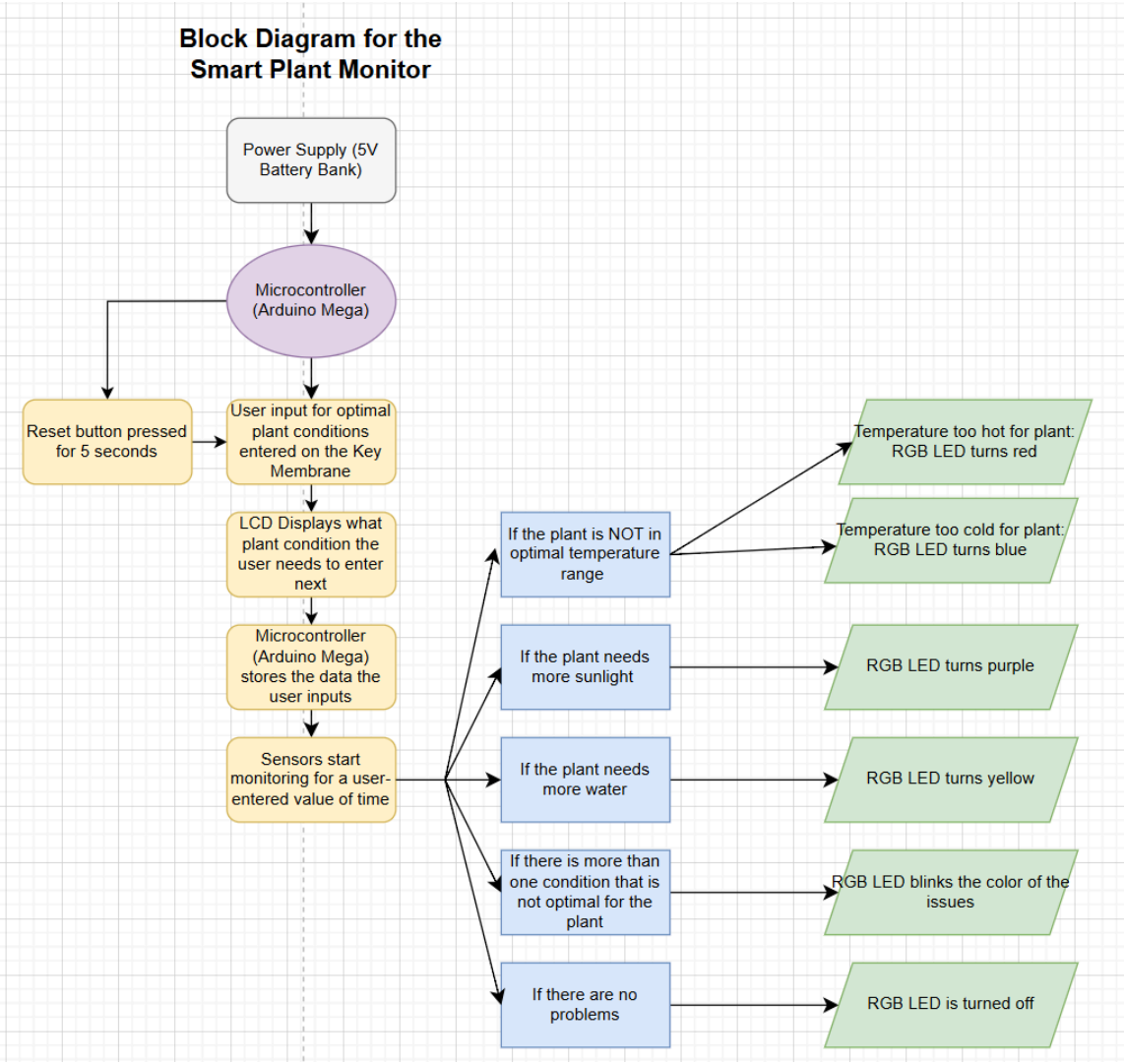

Figure 1: Code Flow Chart

Final Project Code

```
#include "DHT.h"
#include <Keypad.h>
#include <LiquidCrystal.h>

// --- MODE SWITCH ---
#define TEST_MODE          // Uncomment for test mode (seconds instead of hours)
//#define REAL_MODE        // Comment this line for test mode

// Pins
const int soilSensorPin = A0;
const int lightSensorPin = A1;
const int DHTPin = 2;
const int redPin = 9;
const int greenPin = 10;
const int bluePin = 11;
const int resetButtonPin = 7;   // RESET BUTTON PIN

#define DHTTYPE DHT11
DHT dht(DHTPin, DHTTYPE);

// Keypad setup
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {22,23,24,25};
byte colPins[COLS] = {26,27,28,29};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

// LCD setup
```

```
LiquidCrystal lcd(30, 31, 32, 33, 34, 35);

String inputString = "";

// Sensor ranges
int soilLower = 0;
float tempLower = 0;
float tempUpper = 0;
int lightThreshold = 0;

// Nighttime floor for light sensor
const int nightLevel = 50;

// Timing
unsigned long previousMillis = 0;

// --- MODIFIED: interval is now user-defined ---
unsigned long userInterval = 15000;  // user sets this
unsigned long interval = 15000;      // used by the loop

bool firstLoop = true;

// Sunlight tracking
unsigned long sunlightSeconds = 0;
unsigned long darkDuration = 0;
bool sunlightSatisfied = false;

// Adjustable night duration
#ifdef TEST_MODE
const unsigned long nightResetSeconds = 60;
#else
const unsigned long nightResetSeconds = 7200;
#endif

// Sunlight goal
unsigned long sunlightGoalSeconds = 0;
```

```
// --- Variables for reset hold detection ---
unsigned long resetPressStart = 0;
const unsigned long resetHoldTime = 5000; // 5 seconds hold

// --- LED blinking globals ---
const int MAX_ISSUES = 4;                 // Max simultaneous issues
float blinkColors[MAX_ISSUES][3];         // RGB values for each active
issue
int blinkCount = 0;                        // Current color index
const unsigned long ledOnTime = 800;      // LED on duration
const unsigned long ledOffTime = 400;     // LED off duration
bool ledState = false;                     // LED currently ON/OFF
unsigned long ledPreviousMillis = 0;
int activeIssues = 0;                      // Number of active issues

// Flags for active issues
bool highTemp = false;
bool lowTemp = false;
bool drySoil = false;
bool lowLightFlag = false;

// ------------------------
// Check Reset Button
// ------------------------
void checkResetButton() {
  if (digitalRead(resetButtonPin) == LOW) { // Button pressed
    if (resetPressStart == 0) {
      resetPressStart = millis(); // start timing
    } else if (millis() - resetPressStart >= resetHoldTime) {
      Serial.println("\n--- RESET BUTTON HELD FOR 5 SECONDS ---");
      setLED(0,0,0); // turn off LED
      delay(200);
      setup(); // restart system
    }
  } else {
    resetPressStart = 0; // button released, reset timer
  }
```

```
}

// ------------------------
// Keypad Input Function
// ------------------------
int getKeypadValue(String prompt, int maxDigits = 3, String lcdLabel = "")
{
  inputString = "";
  Serial.println();
  Serial.print("🌿 Smart Plant Monitor with RGB LED Alerts (°F) 🌿\n");
  Serial.print(prompt + ": ");

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(lcdLabel);
  lcd.setCursor(0,1);
  lcd.print(inputString);

  char lastKey = NO_KEY;

  while (true) {
    checkResetButton(); // check reset during input

    char key = keypad.getKey();
    if (key && key != lastKey) {
      lastKey = key;

      if (key >= '0' && key <= '9') {
        if (inputString.length() < maxDigits) {
          inputString += key;
          Serial.print(key);
          lcd.setCursor(0,1);
          lcd.print("                ");
          lcd.setCursor(0,1);
          lcd.print(inputString);
        }
      }
```

```
      else if (key == 'B') { // Backspace
        if (inputString.length() > 0) {
          inputString.remove(inputString.length() - 1);
          Serial.println();
          Serial.print(prompt + ": " + inputString);
          lcd.setCursor(0,1);
          lcd.print("                ");
          lcd.setCursor(0,1);
          lcd.print(inputString);
        }
      }
      else if (key == 'C') { // Clear → restart all inputs
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Smart Plant Monitor");
        return -1;
      }
      else if (key == 'A') { // Enter
        if (inputString.length() > 0) {
          int value = inputString.toInt();
          Serial.println();
          Serial.print("✅ You entered: "); Serial.println(value);
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print(lcdLabel + ": " + String(value));
          delay(1000);
          lcd.clear();
          while (keypad.getKey() == 'A') { delay(10); }
          return value;
        }
      }
    }
    else if (!key) {
      lastKey = NO_KEY;
    }
  }
}
```

```
// ------------------------
// Setup
// ------------------------
void setup() {
  resetPressStart = 0; // clear previous press

  Serial.begin(9600);
  pinMode(soilSensorPin, INPUT);
  pinMode(lightSensorPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(resetButtonPin, INPUT_PULLUP);
  dht.begin();
  lcd.begin(16,2);

  // Reset all measurements and LED
  sunlightSeconds = 0;
  darkDuration = 0;
  sunlightSatisfied = false;
  setLED(0,0,0);
  previousMillis = millis();

  while (true) {
    checkResetButton(); // allow reset during setup

    int val = -1;

    // --- Soil ---
    do {
      val = getKeypadValue("Enter minimum soil moisture level
(0-100)",3,"Min Soil %");
      if (val == -1) continue;
      if (val < 0 || val > 100) Serial.println("❌ Soil Level must be
0-100%");
    } while (val < 0 || val > 100);
```

```
    soilLower = val;

    // --- Lower Temp ---
    do {
      val = getKeypadValue("Enter LOWER optimal temperature (°F,
0-150)",3,"Lower Temp");
      if (val == -1) continue;
      if (val < 0 || val > 150) Serial.println("❌ Temp must be 0-150°F");
    } while (val < 0 || val > 150);
    tempLower = val;

    // --- Upper Temp ---
    do {
      val = getKeypadValue("Enter UPPER optimal temperature (°F, >= lower,
<=150)",3,"Upper Temp");
      if (val == -1) continue;
      if (val < tempLower || val > 150) Serial.println("❌ Upper temp must
be ≥ lower and ≤150");
    } while (val < tempLower || val > 150);
    tempUpper = val;

    // --- Light ---
    do {
      val = getKeypadValue("Enter minimum optimal light level
(0-1023)",4,"Min Light 0-1023");
      if (val == -1) continue;
      if (val < 0 || val > 1023) Serial.println("❌ Light must be
0-1023");
    } while (val < 0 || val > 1023);
    lightThreshold = val;

    // --- Sunlight Goal ---
#ifdef TEST_MODE
    do {
      val = getKeypadValue("Enter sunlight goal (seconds, 1-100)",3,"Light
Goal (s)");
      if (val == -1) continue;
```

```
      if (val < 1 || val > 100) Serial.println("❌ Goal must be 1-100
seconds");
    } while (val < 1 || val > 100);
    sunlightGoalSeconds = val;
#else
    do {
      val = getKeypadValue("Enter sunlight goal (hours, 1-12)",3,"Light
Goal (hrs)");
      if (val == -1) continue;
      if (val < 1 || val > 12) Serial.println("❌ Goal must be 1-12
hours");
    } while (val < 1 || val > 12);
    sunlightGoalSeconds = val * 3600UL;
#endif

    // --- Monitoring Interval ---
#ifdef TEST_MODE
    do {
      val = getKeypadValue("Enter monitor interval (seconds, 1-300)", 3,
"Interval (s)");
      if (val == -1) continue;
      if (val < 1 || val > 300) Serial.println("❌ Interval must be 1-300
seconds");
    } while (val < 1 || val > 300);
    userInterval = val * 1000UL;
#else
    do {
      val = getKeypadValue("Enter monitor interval (minutes, 1-60)", 2,
"Interval(min)");
      if (val == -1) continue;
      if (val < 1 || val > 60) Serial.println("❌ Interval must be 1-60
minutes");
    } while (val < 1 || val > 60);
    userInterval = val * 60000UL;
#endif

    interval = userInterval; // apply user interval
```

```
#ifdef TEST_MODE
    Serial.print("📏 Monitoring every ");
    Serial.print(interval / 1000);
    Serial.println(" seconds.");
#else
    Serial.print("📏 Monitoring every ");
    Serial.print(interval / 60000);
    Serial.println(" minutes.");
#endif

    break;
  }

  firstLoop = true;
  previousMillis = millis() - interval;

#ifdef TEST_MODE
  Serial.print("---- START MONITORING EVERY ");
  Serial.print(interval / 1000);
  Serial.println(" SECONDS ----");
#else
  Serial.print("---- START MONITORING EVERY ");
  Serial.print(interval / 60000);
  Serial.println(" MINUTES ----");
#endif
}

// ------------------------
// Helper function to set RGB LED
// ------------------------
void setLED(float r, float g, float b) {
  analogWrite(redPin, (int)(r * 255));
  analogWrite(greenPin, (int)(g * 255));
  analogWrite(bluePin, (int)(b * 255));
}
```

```
// ------------------------
// Main Loop
// ------------------------
void loop() {
  checkResetButton();

  unsigned long currentMillis = millis();

  // --- Measurement Interval Check ---
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    int soilValue = analogRead(soilSensorPin);
    int soilPercent = map(soilValue, 1023, 0, 0, 100);

    float temperatureC = dht.readTemperature();
    float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;

    int lightValue = analogRead(lightSensorPin);

    // --- Sunlight / Dark Duration Logic ---
    if (!firstLoop) {
      if (lightValue > lightThreshold) {
        sunlightSeconds += interval / 1000;
        darkDuration = 0;
      } else if (lightValue < nightLevel) {
        darkDuration += interval / 1000;
      } else {
        darkDuration = 0;
      }

      if (darkDuration >= nightResetSeconds) {
        sunlightSeconds = 0;
        sunlightSatisfied = false;
        darkDuration = 0;
        Serial.println("🌙 Night detected. Resetting daily sunlight
counter.");
```

```
      }

      if (sunlightSeconds >= sunlightGoalSeconds) {
        sunlightSatisfied = true;
        Serial.println("✅ Sunlight goal reached!");
      }
    }

    // --- Print Sensor Readings ---
    Serial.print("🌱 Soil Moisture: "); Serial.print(soilPercent);
Serial.print("% (Optimal ≥ "); Serial.print(soilLower);
Serial.print("%)");
    if (soilPercent < soilLower) Serial.print("  --> TOO DRY!");
    Serial.println();

    Serial.print("🌡️ Temperature: "); Serial.print(temperatureF,2);
Serial.print(" °F (Optimal "); Serial.print(tempLower,2); Serial.print(" -
"); Serial.print(tempUpper,2); Serial.print(" °F)");
    if (temperatureF < tempLower || temperatureF > tempUpper)
Serial.print("  --> OUT OF OPTIMAL TEMP RANGE!");
    Serial.println();

    Serial.print("💡 Light Level: "); Serial.print(lightValue);
Serial.print(" (Optimal ≥ "); Serial.print(lightThreshold);
Serial.print(")");
    if (lightValue < nightLevel) Serial.print(" 🌙 Night detected");
    else if (lightValue < lightThreshold && lightValue > nightLevel &&
!sunlightSatisfied) Serial.print("  --> TOO DARK!");
    Serial.println();

#ifdef TEST_MODE
    Serial.print("☀️ Sunlight accumulated today: ");
Serial.print(sunlightSeconds); Serial.print(" seconds / goal (");
Serial.print(sunlightGoalSeconds); Serial.println(" seconds)");
#else
    Serial.print("☀️ Sunlight accumulated today: ");
Serial.print(sunlightSeconds/3600.0,2); Serial.print(" hrs / goal (");
```

```
Serial.print(sunlightGoalSeconds/3600UL); Serial.println(" hrs)");
#endif

    Serial.println("--------------------------------------");

    // --- Determine active issues ---
    highTemp = (temperatureF > tempUpper);
    lowTemp  = (temperatureF < tempLower);
    drySoil  = (soilPercent < soilLower);
    lowLightFlag = (lightValue < lightThreshold && lightValue > nightLevel
&& !sunlightSatisfied);

    // --- Handle LED based on active issues ---
    activeIssues = 0;
    int idx = 0;

    if (highTemp)  { blinkColors[idx][0]=1; blinkColors[idx][1]=0;
blinkColors[idx][2]=0; idx++; activeIssues++; } // red
    if (lowTemp)   { blinkColors[idx][0]=0; blinkColors[idx][1]=0;
blinkColors[idx][2]=1; idx++; activeIssues++; } // blue
    if (drySoil)   { blinkColors[idx][0]=1; blinkColors[idx][1]=1;
blinkColors[idx][2]=0; idx++; activeIssues++; } // yellow
    if (lowLightFlag) { blinkColors[idx][0]=1; blinkColors[idx][1]=0;
blinkColors[idx][2]=1; idx++; activeIssues++; } // purple

    blinkCount = 0;
    ledState = false;
    ledPreviousMillis = millis();

    // --- Single issue static LED ---
    if (activeIssues == 1) {
      setLED(blinkColors[0][0], blinkColors[0][1], blinkColors[0][2]);
    }
    // --- No issues ---
    if (activeIssues == 0) setLED(0,0,0);

    firstLoop = false;
```

```
  }

  // --- Non-blocking LED blink for multiple issues ---
  if (activeIssues > 1) {
    unsigned long currentMillis = millis();
    if (ledState) {
      if (currentMillis - ledPreviousMillis >= ledOnTime) {
        setLED(0,0,0); // turn off
        ledState = false;
        ledPreviousMillis = currentMillis;
        blinkCount++;
        if (blinkCount >= activeIssues) blinkCount = 0;
      }
    } else {
      if (currentMillis - ledPreviousMillis >= ledOffTime) {
        setLED(blinkColors[blinkCount][0], blinkColors[blinkCount][1],
blinkColors[blinkCount][2]);
        ledState = true;
        ledPreviousMillis = currentMillis;
      }
    }
  }
}
```