```
// Initialize constants for PID control
KP = 35.0        // Proportional constant: adjusts response to the current error
KI = 0.8         // Integral constant: addresses accumulated past errors
KD = 15.0        // Derivative constant: smooths response by predicting future error based on rate of
change
PID_INTERVAL = 100 ms // Interval for executing PID control adjustments

// Temperature control settings and fan operation limits
MIN_FAN_DUTY = 128    // Lowest fan speed duty cycle (half on)
MAX_FAN_DUTY = 255      // Highest fan speed duty cycle (fully on)
MIN_PELTIER_DUTY = 64   // Minimum Peltier duty to prevent excessive wear
MAX_PELTIER_DUTY = 255  // Maximum Peltier duty for max cooling
MIN_TEMP_DIFFERENCE = 0.4   // Minimum difference to activate Peltier cooling
FAN_TEMP_THRESHOLD = 2.0    // Difference needed to set fan to max
FAN_RESPONSE_FACTOR = 1.0   // Scaling factor for fan speed response

// Initialize Display and Rotary Encoder
Initialize OLED display with connection to appropriate pins
Initialize Rotary Encoder on specified pins (e.g., CLK, DT, SW)

// Declare global variables to store control values
currentSetpoint = 20           // Target temperature user wants to maintain
lastError = 0                  // Previous error value for calculating derivative
integral = 0                   // Sum of errors for integral term
lastPID = 0                    // Last timestamp when PID function was run

// 1. Read and Calculate Temperature Function
FUNCTION readTemperature:
    // Initialize array to store multiple samples
    samples = array with NUM_SAMPLES elements

    FOR i = 0 TO NUM_SAMPLES:
        // Read raw analog value from thermistor
        rawValue = analogRead(thermistorPin)

        // Calculate temperature from thermistor value using the Steinhart-Hart equation
        temperature = calculateTemperature(rawValue)

        // Store calculated temperature in the sample array
        samples[i] = temperature

        // Wait briefly before next sample to reduce noise
        Delay SAMPLE_DELAY ms

    // Sort sample array and find the median value to get a stable reading
    medianTemperature = median(samples)
    RETURN medianTemperature

// 2. Calculate Fan Speed Based on Temperature Difference
FUNCTION calculateFanDuty(tempDifference):
    IF tempDifference <= 0:
        RETURN MIN_FAN_DUTY       // Minimum cooling needed
    ELSE:
        // Calculate fan speed proportionally based on temperature difference
        fanDuty = MIN_FAN_DUTY + tempDifference * FAN_RESPONSE_FACTOR
```

```
        fanDuty = constrain(fanDuty, MIN_FAN_DUTY, MAX_FAN_DUTY)
        RETURN fanDuty            // Return constrained fan duty cycle

// 3. Update PWM Duty Cycle on Specified Pin
FUNCTION updatePWM(pin, duty):
    // Update the PWM duty cycle for fan or Peltier based on calculated duty
    IF pin == fanPin:
        analogWrite(fanPin, duty)       // Set fan speed
    ELSE IF pin == peltierPin:
        analogWrite(peltierPin, duty)   // Set Peltier cooling level

// 4. Retrieve Current PWM Duty Cycle of Specified Pin
FUNCTION getPWMDuty(pin):
    RETURN current duty cycle of specified pin (fan or Peltier)

// 5. Core PID Control Function for Temperature Management
FUNCTION updatePID(currentTemp):
    currentTime = get current time in milliseconds
    dt = currentTime - lastPID      // Calculate time elapsed since last PID update

    IF dt < PID_INTERVAL:
        RETURN                      // Skip if not enough time has passed

    // Calculate error between setpoint and current temperature
    error = currentSetpoint - currentTemp

    // Calculate fan speed based on how far temperature is from setpoint
    fanDuty = calculateFanDuty(error)
    updatePWM(fanPin, fanDuty)     // Apply calculated fan speed

    IF abs(error) > MIN_TEMP_DIFFERENCE:
        // Calculate proportional term
        proportional = KP * error

        // Update integral term with accumulated error, applying anti-windup limit
        integral += error * dt
        integral = constrain(integral, -MAX_INTEGRAL, MAX_INTEGRAL)
        integralTerm = KI * integral

        // Calculate derivative term based on rate of error change
        derivative = (error - lastError) / dt
        derivativeTerm = KD * derivative

        // PID output is the sum of proportional, integral, and derivative terms
        output = proportional + integralTerm + derivativeTerm

        // Map output to Peltier’s duty range to prevent under/overpowering
        peltierDuty = map(output, MIN_PELTIER_DUTY, MAX_PELTIER_DUTY)
        updatePWM(peltierPin, peltierDuty) // Apply Peltier cooling power based on PID output
    ELSE:
        // If temperature is close to target, turn Peltier off or minimum duty
        updatePWM(peltierPin, MIN_PELTIER_DUTY)
        integral = 0                // Reset integral to prevent windup

    lastError = error               // Store error for next derivative calculation
```

```
    lastPID = currentTime          // Update last PID run time

// 6. Main Program Loop for Display, Temperature Update, and PID Control
WHILE program is running:
    currentTime = get current time in milliseconds

    IF currentTime - lastDisplayUpdate >= 250 ms:
        // Read filtered current temperature from sensor
        currentTemperature = readTemperature()

        // Call PID function with current temperature to control fan and Peltier
        updatePID(currentTemperature)

        // Update OLED display with target and current temperature
        updateDisplay(currentSetpoint, currentTemperature)

        // Update last display update time
        lastDisplayUpdate = currentTime

    // Small delay for processing cycle
    Delay 25 ms

// 7. Button Click Handler to Update Target Setpoint with Rotary Encoder
FUNCTION rotary_onButtonClick:
    currentTime = get current time in milliseconds

    // Debounce button to prevent multiple trigger within short interval
    IF currentTime - lastButtonPress < 500 ms:
        RETURN

    // Update setpoint based on rotary encoder input (increase/decrease temperature)
    IF encoder rotation detected:
        change = encoder rotation value
        currentSetpoint += change   // Adjust setpoint by encoder step

    // Display updated setpoint on OLED
    displaySetpoint(currentSetpoint)
    lastButtonPress = currentTime   // Update last button press time
```