```
/*EE_TempRegulator_Project.ino
   This program contains the code for a temperature regulator
   which checks the current ambient temperature and adjusts
   the speed of a DC fan accordingly.
   It has two modes: Manual (user sets fan speed)
   and Automatic (user sets desired temperature).
   Sensors:
   - Thermistor (temperature sensor)
   - Potentiometer
   - Push Button
   Actuators:
   - 120mm 4-pin PWM RGB Fan (12V DC motor)
   - 16x2 LCD

   Date: 04/20/2022
   Author: Daniel Noronha & Justin Yatsuhashi
   EE 10115 Temperature Regulator Project Code
*/

// Include LCD header
#include <LiquidCrystal.h>
// Set mode to manual by default but keep it volatile
// Can be accessed by interrupt
volatile bool automatic = false;
// Hardware limitations mean PWM duty cycle < 26 has no effect.
const int min_PWM_speed = 26; // ~10% fan speed (26/255)
// Define hardware pins
const int fan_PWM_pin = 9;
const int button_pin = 2;

LiquidCrystal lcd(11, 12, 4, 5, 6, 7);

void setup() {
  // Setup code which runs on startup only
  // Set fan digital PWM pin mode to OUTPUT
  pinMode(fan_PWM_pin, OUTPUT);
  // Attach rising interrupt to push button (on pin 2 = 0)
  attachInterrupt(button_pin-2, modeSwitch, RISING);
  // Initialize 16x2 LCD
  lcd.begin(16, 2);
  // Display Startup Message for 2.5 seconds
  lcd.setCursor(1,0);
  lcd.print("Temp Regulator");
  lcd.setCursor(0,1);
  lcd.print("By Daniel&Justin");
  delay(2500);
}

void loop() {
  // Main program loop (indefinite)
  // First clear the LCD
  lcd.clear();
  // Read thermistor temperature value (unmapped)
```

```
  int sensorVal = analogRead(A3);
  // Read potentiometer value
  int potVal = analogRead(A0);

  // Use therm2cel function defined below to map thermistor value to
temperature
  float temp = therm2cel(sensorVal);// temperature in °C
  // Display current temperature on first line
  lcd.print(temp);
  lcd.print("*C");
  lcd.print("  ");
  lcd.print(temp*1.8+32); // °C to °F
  lcd.print("*F");
  lcd.setCursor(0,1);
  // If the mode is automatic:
  if (automatic) {
    // Map knob setting to temperature range (21°C-27°C)
    int settemp = map(potVal, 0, 1023, 21, 27);
    // Display set temperature on second line
    lcd.print("Set: ");
    lcd.print(settemp);
    lcd.print("*C=");
    lcd.print(int(settemp*1.8+32)); // °C to °F
    lcd.print("*F A"); // Show selected mode as Automatic (A)
    // If the current temperature is less than 1°C above the
    // set temperature, then set fan speed to 50% (127/255)
    if (int(temp) == settemp) analogWrite(fan_PWM_pin, 127);
    // Otherwise, if the current temperature is more than 1°C above the
    // set temperature, then set fan speed to 100% (255/255)
    else if (temp > settemp) analogWrite(fan_PWM_pin, 255);
    // If the current temperature is less than the set temperature,
    // then set the fan to idle (10%)
    else analogWrite(fan_PWM_pin, 0);
  }

  else {
    // Manual Mode
    // Map knob setting to PWM fan speed (26-255)
    int fanSpeed = map(potVal, 0, 1023, min_PWM_speed, 255);
    // Express PWM fan speed as a percentage (10%-100%)
    int percent = (fanSpeed/255.0) * 100;
    // Send PWM signal to fan
    analogWrite(fan_PWM_pin,fanSpeed);
    // Display user-set fan speed percentage on second line
    lcd.print(percent);
    lcd.print("%");
    lcd.setCursor(15,1);
    lcd.print("M"); // Show selected mode as Manual (M)
  }
  delay(500);
  // Wait half a second to let user read display before clearing

}
```

```
106
107 float therm2cel(int thermval) {
108   // User-defined function to map thermistor value to temperature (°C)
109   // Map thermistor value to millivolts (0-5mV)
110   float mVolts = map(thermval, 0, 1023, 0, 5000);
111   // Use linear equation to convert voltage to Celsius temperature
112   float temp = (mVolts - 500.0) / 10.0;
113   return temp;
114 }
115
116 void modeSwitch() {
117   // Interrupt Service Routine to toggle system mode selection (A/M)
118   // Checks how long it has been since last processed interrupt
119   // If it has been less than 0.2s, do not process interrupt
120   // Eliminates button bounce
121   static unsigned long last_processed_interrupt_time = 0;
122   unsigned long current_interrupt_time = millis();
123   if ((current_interrupt_time - last_processed_interrupt_time) > 200) {
124       // Bit-flip boolean value of global volatile variable automatic (0/1)
125       automatic = !automatic;// false=>true or true=>false
126   }
127   last_processed_interrupt_time = current_interrupt_time;
128 }
129
130 // End of Program //
```