```
// ============================================
// 🎯 ¡MÁQUINA RECICLADORA DE BOTELLAS DE PLÁSTICO!
// ============================================
// Este programa controla una máquina que transforma botellas de plástico PET
// en "hilo" para impresoras 3D. ¡Es como una máquina de hacer fideos, pero con plástico!
//
// La máquina tiene 2 partes principales:
// 1. Un MOTOR que tira del plástico derretido (como tirar masa de pizza)
// 2. Un CALEFACTOR que derrite el plástico a 240°C (como un horno)
//
// El Arduino es el "cerebro" que controla todo automáticamente.
// ============================================


// ============================================
// PASO 1: INCLUIR LIBRERÍAS
// ============================================
// Las librerías son como "cajas de herramientas" que otras personas ya hicieron
// y nosotros podemos usar. Nos ahorran mucho trabajo.

#include <Wire.h>               // Esta librería permite hablar con el display por I2C
                                // I2C es como un "idioma" que usa solo 2 cables para
comunicarse

#include <LiquidCrystal_I2C.h>  // Esta librería controla el display LCD (la pantallita)
                                // LCD = Liquid Crystal Display (pantalla de cristal
líquido)


// ============================================
// PASO 2: CONFIGURAR EL DISPLAY LCD
// ============================================
// Creamos un objeto llamado "lcd" que representa nuestra pantalla.
// Es como ponerle un nombre a la pantalla para poder hablarle.

LiquidCrystal_I2C lcd(0x27, 16, 2);  // 0x27 es la "dirección" del display (como una
dirección de casa)
                                     // 16 = tiene 16 columnas (16 letras por fila)
                                     // 2 = tiene 2 filas (2 líneas de texto)
                                     // Si no funciona, prueba cambiar 0x27 por 0x3F


// ============================================
// PASO 3: DEFINIR LOS PINES DEL MOTOR
// ============================================
// Los "pines" son como los enchufes del Arduino donde conectamos cables.
// Cada pin tiene un número, como si fueran "apartamentos" numerados.

const int dirPin = 2;    // Pin número 2: controla la DIRECCIÓN del motor
                         // Le dice al motor si debe girar hacia adelante o hacia atrás
                         // "const" significa que este número NUNCA va a cambiar

const int stepPin = 3;   // Pin número 3: controla los PASOS del motor
                         // Cada vez que este pin cambia, el motor da "un pasito"
                         // Es como cuando caminas: cada paso te mueve un poquito

// --- PINES DE MICROSTEPPING (¡NUEVO! Para eliminar vibración) ---
```

```
// Estos 3 pines controlan cuán "suave" se mueve el motor.
// MS1, MS2, MS3 = MicroStep 1, 2, 3
// Los ponemos en HIGH para activar modo 1/16 (súper suave)

const int ms1Pin = 8;    // Pin 8: conectado a MS1 del A4988
const int ms2Pin = 12;   // Pin 12: conectado a MS2 del A4988
const int ms3Pin = 13;   // Pin 13: conectado a MS3 del A4988

// Tabla de microstepping del A4988:
// MS1  MS2  MS3  →  Modo
// LOW  LOW  LOW  →  Full step (1/1)   ← modo viejo (mucha vibración)
// HIGH HIGH HIGH →  1/16 step          ← modo nuevo (súper suave)


// ==========================================
// PASO 4: DEFINIR EL PIN DEL POTENCIÓMETRO
// ==========================================
// El potenciómetro es la "perilla giratoria" para controlar la velocidad.
// Es como el acelerador de un auto o el control de volumen de la radio.

const int potPin = A0;   // Pin A0 (es un pin "analógico", lee valores de 0 a 1023)
                         // Cuando giramos la perilla, este pin lee un número diferente


// ==========================================
// PASO 5: VARIABLES PARA EL MOTOR (INTERRUPCIONES)
// ==========================================
// Estas variables controlan cómo funciona el motor.
// "volatile" es una palabra especial que significa: "esta variable puede cambiar
// en cualquier momento por una interrupción, ¡así que no la optimices!"

volatile unsigned int motorDelayMicros = 5000;  // Tiempo entre cada paso del motor (en
microsegundos)
                                                // Cuanto MÁS PEQUEÑO el número = motor
MÁS RÁPIDO
                                                // 5000 microsegundos = 0.005 segundos

volatile bool motorHabilitado = true;   // "bool" = variable que solo puede ser true
(verdadero) o false (falso)
                                        // true = motor ENCENDIDO, false = motor APAGADO

// --- VARIABLES PARA ACELERACIÓN SUAVE (¡NUEVO! Para evitar sacudidas) ---
// En vez de cambiar la velocidad instantáneamente, la cambiamos gradualmente.
// Es como acelerar un auto suavemente en vez de pisar a fondo de golpe.

volatile unsigned int motorDelayObjetivo = 5000;  // Velocidad a la que QUEREMOS llegar
volatile unsigned int motorDelayActual = 5000;    // Velocidad ACTUAL del motor
const unsigned int pasoAceleracion = 50;          // Qué tan rápido acelera (50 = suave)
                                                  // Valor más chico = aceleración más
suave
                                                  // Valor más grande = aceleración más
brusca


// ==========================================
// PASO 6: DEFINIR PINES Y VALORES DEL TERMISTOR
// ==========================================
// El termistor es un "termómetro electrónico" que mide la temperatura.
```

```
// Es como un sensor mágico que cambia su resistencia cuando cambia la temperatura.

const int termistorPin = A1;              // Pin A1: aquí leemos la temperatura

const float R_FIJA = 4700.0;              // Resistencia fija de 4700 ohmios (4.7k)
                                          // Esta resistencia está conectada en serie con
el termistor
                                          // "float" = número decimal (puede tener comas,
como 4.5)

const float R_TERMISTOR_25C = 100000.0;   // A 25°C, el termistor tiene 100,000 ohmios
(100k)
                                          // Es como su "valor de referencia"

const float BETA = 3950.0;                // Número mágico del termistor (dato del
fabricante)
                                          // Se usa en la fórmula matemática para calcular
temperatura


// ==========================================
// PASO 7: DEFINIR PIN Y VARIABLES DEL CALEFACTOR
// ==========================================
// El MOSFET es como un "interruptor electrónico" que controla el calefactor.
// Puede apagarlo, encenderlo, o ponerlo a "media potencia".

const int mosfetPin = 11;                // Pin 11: controla la potencia del calefactor
                                         // Usa PWM (Pulse Width Modulation) = control de
potencia variable
                                         // Es como un interruptor que prende y apaga SUPER
rápido

float tempObjetivo = 240.0;              // Temperatura que QUEREMOS alcanzar (en grados
Celsius)
                                         // 240°C es perfecto para derretir PET (plástico
de botellas)

bool controlCalefactorActivo = false;    // ¿Está el control automático ENCENDIDO?
                                         // false = apagado, true = encendido


// ==========================================
// PASO 8: VARIABLES DEL CONTROL PID
// ==========================================
// PID es un "piloto automático inteligente" que controla la temperatura.
// Es como el control de crucero de un auto, pero para temperatura.
// PID significa: Proporcional, Integral, Derivativo (son 3 tipos de correcciones)

float Kp = 8.0;     // "Proporcional": cuánto reaccionar según QUÉ TAN LEJOS estamos del
objetivo
                    // Si estamos MUY lejos = corregir MUCHO
                    // Si estamos cerca = corregir poquito

float Ki = 0.05;    // "Integral": corregir errores que SE ACUMULAN con el tiempo
                    // Es como compensar por quedarse siempre un poquito abajo del
objetivo

float Kd = 120.0;   // "Derivativo": ANTICIPAR y frenar ANTES de llegar al objetivo
```

```
                    // Es como frenar el auto ANTES de llegar al semáforo
                    // ¡Evita que nos "pasemos" de temperatura!

float errorAnterior = 0.0;    // Guardamos cuál era el error la última vez (para calcular
"D")
float errorAcumulado = 0.0;   // Guardamos la SUMA de todos los errores (para calcular
"I")


// ==========================================
// PASO 9: DEFINIR PINES DE LOS BOTONES
// ==========================================
// Los botones son como "interruptores" que presionamos con el dedo.

const int botonOnOff = 5;   // Pin 5: botón para PRENDER/APAGAR el control de temperatura
const int botonBajar = 6;   // Pin 6: botón para BAJAR la temperatura objetivo (-5°C)
const int botonSubir = 7;   // Pin 7: botón para SUBIR la temperatura objetivo (+5°C)

const int botonInversion = 4;   // Pin 4: botón para cambiar la DIRECCIÓN del motor
                                // Hace que el motor gire al revés

bool direccionActual = LOW;     // Guardamos en qué dirección va el motor por defecto
                                // LOW = dirección por defecto, HIGH = invertida


// ==========================================
// PASO 10: VARIABLE PARA LAS INTERRUPCIONES
// ==========================================
// Esta variable se usa en la "interrupción" que mueve el motor.
// Una interrupción es como una "alarma" que suena automáticamente.

volatile bool pulsoAlto = false;  // ¿El pulso del motor está en HIGH o LOW?
                                  // El motor necesita que este valor alterne: HIGH, LOW,
HIGH, LOW...


// ==========================================
// PASO 11: FUNCIÓN DE INTERRUPCIÓN (ISR)
// ==========================================
// ISR = Interrupt Service Routine = "Rutina de Servicio de Interrupción"
// Esta función se ejecuta AUTOMÁTICAMENTE cada cierto tiempo, como un despertador.
// NO la llamamos nosotros, ¡el Timer1 la llama solito!
//
// ¿Para qué sirve? Para mover el motor SIN BLOQUEAR el resto del programa.
// Es como poder caminar Y masticar chicle al mismo tiempo.

ISR(TIMER1_COMPA_vect) {
  // Esta función se ejecuta automáticamente cada pocos microsegundos

  if (motorHabilitado) {  // Solo si el motor está habilitado...

    // --- ACELERACIÓN SUAVE (¡NUEVO!) ---
    // Cada vez que se ejecuta la ISR, acercamos la velocidad actual a la velocidad
objetivo.
    // Es como acelerar gradualmente un auto en vez de pisar a fondo de golpe.

    if (motorDelayActual > motorDelayObjetivo) {
      // Si vamos más LENTO de lo que queremos → ACELERAR
```

```
      // (delay más grande = velocidad más lenta)
      motorDelayActual -= pasoAceleracion;  // Reducir el delay = ir más rápido
      if (motorDelayActual < motorDelayObjetivo) {
        motorDelayActual = motorDelayObjetivo;  // No pasarse del objetivo
      }
      OCR1A = motorDelayActual * 2;  // Actualizar la velocidad del timer

    } else if (motorDelayActual < motorDelayObjetivo) {
      // Si vamos más RÁPIDO de lo que queremos → DESACELERAR
      motorDelayActual += pasoAceleracion;  // Aumentar el delay = ir más lento
      if (motorDelayActual > motorDelayObjetivo) {
        motorDelayActual = motorDelayObjetivo;  // No pasarse del objetivo
      }
      OCR1A = motorDelayActual * 2;  // Actualizar la velocidad del timer
    }

    // --- GENERAR PULSO PARA EL MOTOR ---
    // Alternamos el pin del motor entre HIGH y LOW
    // Es como prender y apagar una luz super rápido: PRENDIDO-APAGADO-PRENDIDO-APAGADO
    // Cada vez que cambia, el motor da "un pasito"

    if (pulsoAlto) {
      // Si el pulso está en HIGH, lo bajamos a LOW
      digitalWrite(stepPin, LOW);   // Apagar el pin
      pulsoAlto = false;            // Recordar que ahora está en LOW
    } else {
      // Si el pulso está en LOW, lo subimos a HIGH
      digitalWrite(stepPin, HIGH);  // Encender el pin
      pulsoAlto = true;             // Recordar que ahora está en HIGH
    }
  }
}


// ==========================================
// PASO 12: FUNCIÓN SETUP (SE EJECUTA UNA SOLA VEZ)
// ==========================================
// La función "setup" es como el "despertarse por la mañana" del Arduino.
// Se ejecuta UNA SOLA VEZ cuando enciendes el Arduino.
// Aquí configuramos TODO antes de empezar a trabajar.

void setup() {

  // --- PASO 12.1: ENCENDER Y CONFIGURAR EL DISPLAY ---
  lcd.init();       // "init" = inicializar = "despertá, pantalla!"
  lcd.backlight();  // Encender la luz de fondo (para poder ver las letras)

  // Mostrar mensaje de bienvenida
  lcd.setCursor(0, 0);           // Ir a la columna 0, fila 0 (esquina superior
izquierda)
  lcd.print("Recicladora PET");   // Escribir texto en el display
  lcd.setCursor(0, 1);           // Ir a la columna 0, fila 1 (segunda línea)
  lcd.print("Iniciando...");      // Escribir texto en el display

  delay(1500);   // Esperar 1.5 segundos (1500 milisegundos)
                 // Para que podamos leer el mensaje

  lcd.clear();   // Borrar todo el display (dejarlo en blanco)
```

```
// --- PASO 12.2: CONFIGURAR LOS PINES DEL MOTOR ---
// "pinMode" le dice al Arduino: "este pin va a DAR órdenes (OUTPUT)"

pinMode(dirPin, OUTPUT);    // Pin 2 es SALIDA (nosotros mandamos señales)
pinMode(stepPin, OUTPUT);   // Pin 3 es SALIDA (nosotros mandamos señales)

// ¡NUEVO! Configurar pines de microstepping
pinMode(ms1Pin, OUTPUT);    // Pin 8 es SALIDA
pinMode(ms2Pin, OUTPUT);    // Pin 12 es SALIDA
pinMode(ms3Pin, OUTPUT);    // Pin 13 es SALIDA

// Activar modo 1/16 microstepping (todos en HIGH)
// Esto hace que el motor se mueva 16 veces más suave
digitalWrite(ms1Pin, HIGH);   // MS1 = HIGH
digitalWrite(ms2Pin, HIGH);   // MS2 = HIGH
digitalWrite(ms3Pin, HIGH);   // MS3 = HIGH
// ¡Ahora el motor dará 3200 pasos por vuelta en vez de 200!
// Cada paso es 16 veces más pequeño = movimiento SÚPER suave


// --- PASO 12.3: CONFIGURAR EL PIN DEL CALEFACTOR ---
pinMode(mosfetPin, OUTPUT);       // Pin 11 es SALIDA
digitalWrite(mosfetPin, LOW);     // Apagarlo al inicio (por seguridad)
                                  // LOW = 0 voltios = APAGADO


// --- PASO 12.4: CONFIGURAR LOS PINES DE LOS BOTONES ---
// "INPUT_PULLUP" significa: "este pin va a ESCUCHAR (INPUT) y además
// tiene una resistencia interna activada que lo mantiene en HIGH cuando
// no está presionado"
//
// Cuando presionamos el botón, el pin baja a LOW.
// Cuando soltamos el botón, el pin vuelve a HIGH.

pinMode(botonInversion, INPUT_PULLUP);  // Escuchar el botón de inversión
pinMode(botonOnOff, INPUT_PULLUP);      // Escuchar el botón ON/OFF
pinMode(botonBajar, INPUT_PULLUP);      // Escuchar el botón bajar
pinMode(botonSubir, INPUT_PULLUP);      // Escuchar el botón subir


// --- PASO 12.5: PONER EL MOTOR EN DIRECCIÓN "ADELANTE" ---
digitalWrite(dirPin, direccionActual);  // Escribir HIGH en el pin de dirección
                                        // HIGH = adelante, LOW = atrás


// --- PASO 12.6: CONFIGURAR EL TIMER1 ---
// El Timer1 es como un "reloj interno" del Arduino que cuenta el tiempo.
// Lo vamos a configurar para que genere una "alarma" cada pocos microsegundos.
// Cada vez que suena la alarma, se ejecuta la función ISR que mueve el motor.
//
// ¿Por qué hacemos esto? Porque así el motor se mueve SOLITO en segundo plano,
// y el Arduino puede hacer otras cosas al mismo tiempo (leer botones, actualizar
// el display, controlar la temperatura, etc.)

noInterrupts();  // Apagar TODAS las interrupciones temporalmente
                 // Es como decir "shhh, necesito concentrarme un momento"
```

```
  // Poner los registros del Timer1 en cero (resetear todo)
  TCCR1A = 0;      // Timer Counter Control Register 1 A = 0
  TCCR1B = 0;      // Timer Counter Control Register 1 B = 0
  TCNT1 = 0;       // Timer Counter 1 = 0 (el contador empieza en cero)

  // Configurar el Timer1 en modo CTC (Clear Timer on Compare Match)
  // En este modo, el timer cuenta hasta un número que nosotros elegimos,
  // y cuando llega a ese número, se "resetea" a cero y genera una interrupción.
  TCCR1B |= (1 << WGM12);  // Activar modo CTC
                           // (Esto es configuración avanzada, no te preocupes por
entenderlo)

  // Configurar el "prescaler" del timer
  // El prescaler es como un "divisor de velocidad".
  // El Arduino funciona a 16 MHz (16 millones de pulsos por segundo).
  // Con prescaler de 8, el timer cuenta cada 0.5 microsegundos.
  TCCR1B |= (1 << CS11);   // Prescaler = 8

  // Configurar el valor de comparación
  // Cuando el timer llegue a este número, generará una interrupción.
  OCR1A = 10000;  // Output Compare Register 1 A = 10000
                  // Con prescaler 8, esto equivale a 5000 microsegundos (5 milisegundos)
                  // Cada 5ms sonará la "alarma" y se moverá el motor

  // Activar la interrupción por comparación
  TIMSK1 |= (1 << OCIE1A);  // Timer Interrupt Mask Register 1
                            // Esto le dice al Arduino: "cuando el timer llegue a OCR1A,
                            // ejecutá automáticamente la función ISR"

  interrupts();  // Volver a encender todas las interrupciones
                 // Es como decir "ok, ya terminé de configurar, podés seguir"
}


// ================================================
// PASO 13: FUNCIÓN LOOP (SE EJECUTA CONSTANTEMENTE)
// ================================================
// La función "loop" es como el "latido del corazón" del Arduino.
// Se ejecuta una y otra y otra vez, miles de veces por segundo, PARA SIEMPRE.
//
// Mientras el Arduino esté encendido, esta función se repite infinitamente:
// loop() -> loop() -> loop() -> loop() -> loop() -> ...
//
// Aquí ponemos todo lo que el Arduino tiene que hacer continuamente:
// - Leer botones
// - Leer el potenciómetro
// - Medir temperatura
// - Controlar el calefactor
// - Actualizar el display

void loop() {

  // --- VARIABLES "STATIC" ---
  // "static" significa: "esta variable MANTIENE su valor entre cada vuelta del loop"
  // Normalmente, las variables se "olvidan" cuando termina la función.
  // Pero las variables "static" son como tener "buena memoria":
  // recuerdan su valor la próxima vez que se ejecuta la función.
```

```
  static unsigned long tiempoUltimoCambioMotor = 0;  // Guarda CUÁNDO fue el último cambio de velocidad/dirección
                                                     // "unsigned long" = número entero positivo MUY grande
                                                     // Sirve para guardar tiempos (milisegundos)


  // ==========================================
  // SECCIÓN 1: LEER LOS BOTONES
  // ==========================================
  // Vamos a leer los 4 botones, pero NO todo el tiempo.
  // Los leeremos solo cada 50 milisegundos (20 veces por segundo).
  // ¿Por qué? Para no "molestar" al programa leyendo todo el tiempo.

  static unsigned long tiempoUltimaLecturaBotones = 0;  // Guarda CUÁNDO fue la última vez que leímos los botones

  // Variables para recordar el estado ANTERIOR de cada botón
  // Necesitamos saber si el botón cambió de NO PRESIONADO a PRESIONADO
  static bool botonInversionAnterior = HIGH;  // HIGH = no presionado
  static bool botonOnOffAnterior = HIGH;
  static bool botonBajarAnterior = HIGH;
  static bool botonSubirAnterior = HIGH;

  // Pregunta: ¿Ya pasaron 50 milisegundos desde la última lectura?
  if (millis() - tiempoUltimaLecturaBotones > 50) {
    // millis() = número de milisegundos desde que se encendió el Arduino
    // Es como un cronómetro que nunca se detiene

    tiempoUltimaLecturaBotones = millis();  // Actualizar el tiempo de la última lectura



    // --- BOTÓN DE INVERSIÓN (cambiar dirección del motor) ---
    bool botonInversionActual = digitalRead(botonInversion);  // Leer el estado ACTUAL del botón
                                                              // HIGH = no presionado, LOW = presionado

    // Detectar si el botón cambió de "no presionado" a "presionado"
    if (botonInversionAnterior == HIGH && botonInversionActual == LOW) {
      // ¡El botón fue PRESIONADO!

      direccionActual = !direccionActual;  // Invertir la dirección
                                           // ! significa "lo contrario"
                                           // Si era HIGH (adelante), ahora es LOW (atrás)
                                           // Si era LOW (atrás), ahora es HIGH (adelante)

      digitalWrite(dirPin, direccionActual);  // Escribir la nueva dirección al motor

      tiempoUltimoCambioMotor = millis();  // Guardar el momento en que cambió
                                           // Esto hará que el display muestre velocidad/dirección por 3 segundos
    }
    botonInversionAnterior = botonInversionActual;  // Recordar el estado actual para la próxima vez
```

```
    // --- BOTÓN ON/OFF DEL CALEFACTOR ---
    bool botonOnOffActual = digitalRead(botonOnOff);  // Leer el botón

    if (botonOnOffAnterior == HIGH && botonOnOffActual == LOW) {
      // ¡El botón fue PRESIONADO!

      controlCalefactorActivo = !controlCalefactorActivo;  // Invertir: si estaba ON → 
OFF, si estaba OFF → ON

      if (!controlCalefactorActivo) {
        // Si acabamos de APAGAR el control...
        digitalWrite(mosfetPin, LOW);  // Apagar el calefactor inmediatamente (por 
seguridad)
      }
    }
    botonOnOffAnterior = botonOnOffActual;  // Recordar el estado


    // --- BOTÓN BAJAR TEMPERATURA ---
    bool botonBajarActual = digitalRead(botonBajar);  // Leer el botón

    if (botonBajarAnterior == HIGH && botonBajarActual == LOW) {
      // ¡El botón fue PRESIONADO!

      tempObjetivo -= 5.0;  // Restar 5 grados a la temperatura objetivo
                            // -= significa "restar y guardar el resultado"
                            // Es lo mismo que: tempObjetivo = tempObjetivo - 5.0

      if (tempObjetivo < 0) {
        // Si la temperatura quedó negativa, ponerla en cero
        // (no queremos temperaturas negativas)
        tempObjetivo = 0;
      }
    }
    botonBajarAnterior = botonBajarActual;  // Recordar el estado


    // --- BOTÓN SUBIR TEMPERATURA ---
    bool botonSubirActual = digitalRead(botonSubir);  // Leer el botón

    if (botonSubirAnterior == HIGH && botonSubirActual == LOW) {
      // ¡El botón fue PRESIONADO!

      tempObjetivo += 5.0;  // Sumar 5 grados a la temperatura objetivo
                            // += significa "sumar y guardar el resultado"

      if (tempObjetivo > 270) {
        // Si la temperatura quedó muy alta, limitarla a 270°C
        // (por seguridad, no queremos más de 270°C)
        tempObjetivo = 270;
      }
    }
    botonSubirAnterior = botonSubirActual;  // Recordar el estado
  }


  // ==========================================
```

```
// SECCIÓN 2: LEER EL POTENCIÓMETRO Y CONTROLAR LA VELOCIDAD
// ==========================================
// El potenciómetro (perilla giratoria) controla la velocidad del motor.
// Lo leemos solo cada 50 milisegundos (igual que los botones).

static unsigned long tiempoUltimaLecturaPot = 0;  // Cuándo fue la última lectura
static int valorPot = 512;        // Valor actual del potenciómetro (0 a 1023)
                                  // 512 es el "punto medio"
static int delayMotor = 5000;     // Tiempo entre pasos del motor (en microsegundos)
static bool motorApagado = false; // ¿Está el motor apagado?

// Pregunta: ¿Ya pasaron 50 milisegundos?
if (millis() - tiempoUltimaLecturaPot > 50) {
  tiempoUltimaLecturaPot = millis();  // Actualizar el tiempo

  // Leemos el potenciómetro VARIAS VECES y promediamos
  // ¿Por qué? Porque las lecturas pueden "saltar" un poco (tener ruido eléctrico)
  // Al promediar varias lecturas, obtenemos un valor más estable y confiable
  int suma = 0;  // Variable para guardar la suma de todas las lecturas

  for (int i = 0; i < 5; i++) {
    // Este "for" se repite 5 veces
    // i empieza en 0, luego 1, luego 2, luego 3, luego 4, y ahí termina

    suma += analogRead(potPin);  // Leer el potenciómetro y sumar al total
                                 // analogRead() lee un valor entre 0 y 1023
                                 // 0 = perilla totalmente a la izquierda
                                 // 1023 = perilla totalmente a la derecha

    delayMicroseconds(100);  // Esperar 100 microsegundos entre lecturas
                             // Es una pausa muy pequeñita
  }

  valorPot = suma / 5;  // Calcular el promedio (dividir la suma entre 5)
                        // Esto nos da un valor más estable


  // Ahora convertimos el valor del potenciómetro en velocidad del motor
  // NOTA: Los rangos están ajustados para microstepping 1/16
  // Con 1/16, el motor necesita dar 16 veces más pasos para la misma velocidad física

  if (valorPot < 20) {
    // Si el potenciómetro está casi en cero (menos de 2%)...
    // APAGAR el motor
    motorApagado = true;
    motorHabilitado = false;  // Desactivar el motor (la ISR ya no lo moverá)

  } else {
    // Si el potenciómetro tiene algún valor...
    // ENCENDER el motor
    motorApagado = false;
    motorHabilitado = true;  // Activar el motor

    // Calcular el delay (tiempo entre pasos) según el valor del potenciómetro
    // Los valores están AJUSTADOS para microstepping 1/16
    // Ahora los delays son más cortos porque necesitamos más pasos

    if (valorPot < 400) {
```

```
        // Rango bajo (20 a 400): velocidades lentas
        delayMotor = map(valorPot, 20, 400, 3000, 1500);
        // AJUSTADO: valores más bajos que antes para compensar microstepping
        // Si valorPot es 20 → delayMotor será 3000 (lento pero suave)
        // Si valorPot es 400 → delayMotor será 1500

      } else if (valorPot > 600) {
        // Rango alto (600 a 1023): velocidades rápidas
        delayMotor = map(valorPot, 600, 1023, 1000, 200);
        // AJUSTADO: valores más bajos para velocidades más altas
        // Si valorPot es 600 → delayMotor será 1000
        // Si valorPot es 1023 → delayMotor será 200 (rápido y suave)

      } else {
        // Rango medio (400 a 600): velocidad media fija
        delayMotor = 1200;  // AJUSTADO: velocidad media para 1/16
      }

      // Actualizar la velocidad OBJETIVO (no la actual)
      // La ISR se encargará de acelerar/desacelerar gradualmente
      motorDelayObjetivo = delayMotor;  // ¡CAMBIO IMPORTANTE! Ahora usamos aceleración
suave
    }
  }


  // ============================================
  // SECCIÓN 3: MEDIR TEMPERATURA Y CONTROLAR EL CALEFACTOR
  // ============================================
  // Aquí leemos el termistor, calculamos la temperatura,
  // ejecutamos el algoritmo PID, y actualizamos el display.
  // Todo esto se hace cada 200 milisegundos (5 veces por segundo).

  static float temperaturaActual = 0.0;  // Temperatura medida (en °C)
  static int valorPotAnterior = 0;       // Valor anterior del potenciómetro (para
detectar cambios)
  static unsigned long tiempoUltimaActualizacionDisplay = 0;  // Cuándo fue la última
actualización

  // Pregunta: ¿Ya pasaron 200 milisegundos?
  if (millis() - tiempoUltimaActualizacionDisplay > 200) {
    tiempoUltimaActualizacionDisplay = millis();  // Actualizar el tiempo


    // --- PASO 3.1: LEER EL TERMISTOR Y CALCULAR TEMPERATURA ---
    // Leemos el termistor MUCHAS VECES (10 veces) y promediamos.
    // ¿Por qué? Para que la lectura sea MUY estable y no "salte".

    float sumaTemperaturas = 0;  // Variable para sumar todas las temperaturas

    for (int i = 0; i < 10; i++) {
      // Este "for" se repite 10 veces

      int lecturaADC = analogRead(termistorPin);  // Leer el pin del termistor
                                                  // Obtenemos un número entre 0 y 1023

      // CALCULAR LA RESISTENCIA DEL TERMISTOR
      // Usamos la fórmula del divisor de tensión:
```

```
  // El circuito es: 5V → R_FIJA (4.7k) → [punto medio] → TERMISTOR → GND
  // En el "punto medio" es donde medimos (pin A1)
  //
  // Fórmula: R_termistor = R_fija × (ADC / (1023 - ADC))
  float resistenciaTermistor = R_FIJA * lecturaADC / (1023.0 - lecturaADC);

  // CALCULAR TEMPERATURA USANDO LA ECUACIÓN DE STEINHART-HART
  // Esta es una fórmula matemática que convierte resistencia en temperatura.
  // Es un poco complicada, pero funciona muy bien.
  //
  // La fórmula completa es:
  // 1/T = 1/T0 + (1/B) × ln(R/R0)
  // Donde:
  // - T = temperatura en Kelvin (la que queremos calcular)
  // - T0 = 298.15 K (que es 25°C en Kelvin)
  // - B = 3950 (constante BETA del termistor)
  // - R = resistencia actual del termistor
  // - R0 = 100000 ohmios (resistencia del termistor a 25°C)
  // - ln = logaritmo natural

  float steinhart;
  steinhart = resistenciaTermistor / R_TERMISTOR_25C;  // R / R0
  steinhart = log(steinhart);                          // ln(R / R0)
  steinhart /= BETA;                                   // (1/B) × ln(R/R0)
  steinhart += 1.0 / 298.15;                           // + 1/T0
  steinhart = 1.0 / steinhart;                         // Invertir para obtener T

  float tempLectura = steinhart - 273.15;  // Convertir de Kelvin a Celsius
                                           // (Celsius = Kelvin - 273.15)

  sumaTemperaturas += tempLectura;  // Sumar esta lectura al total

  delayMicroseconds(100);  // Pequeña pausa entre lecturas
}

// Calcular el PROMEDIO de las 10 lecturas
float tempInstantanea = sumaTemperaturas / 10.0;  // Dividir la suma entre 10


// FILTRO ADICIONAL: SUAVIZADO EXPONENCIAL
// Esto hace que la temperatura cambie "suavemente" en el display,
// en vez de saltar de un número a otro.
// Es como mezclar la temperatura nueva con la anterior:
// 80% de la anterior + 20% de la nueva = cambio gradual

if (temperaturaActual == 0.0) {
  // Si es la primera lectura, usar el valor directamente
  temperaturaActual = tempInstantanea;
} else {
  // Si no es la primera lectura, mezclar con la anterior
  temperaturaActual = (temperaturaActual * 0.8) + (tempInstantanea * 0.2);
}


// --- PASO 3.2: CONTROL PID DEL CALEFACTOR ---
// El PID es como un "piloto automático" que controla la temperatura.
// Calcula cuánta potencia darle al calefactor para llegar exactamente
// a la temperatura objetivo, sin pasarse ni quedarse corto.
```

```
    static int potenciaPWM = 0;           // Potencia que le daremos al calefactor (0 a
255)
    static bool calefactorEncendido = false;  // ¿Está el calefactor encendido?

    if (controlCalefactorActivo) {
      // Solo si el control está ACTIVADO (botón ON)...

      // Calcular el ERROR
      // Error = qué tan lejos estamos del objetivo
      // Si estamos a 200°C y queremos 240°C, el error es +40
      // Si estamos a 250°C y queremos 240°C, el error es -10
      float error = tempObjetivo - temperaturaActual;


      // --- COMPONENTE P (PROPORCIONAL) ---
      // Cuanto más LEJOS estamos del objetivo, más potencia aplicamos.
      // Es como pisar más el acelerador cuando estás más lejos de tu destino.
      float P = Kp * error;
      // Kp = 8.0, entonces si el error es 10°C → P = 80


      // --- COMPONENTE I (INTEGRAL) ---
      // Sumamos todos los errores que tuvimos en el tiempo.
      // Esto corrige errores "persistentes" (que se mantienen mucho tiempo).
      // Es como compensar por siempre quedarnos un poquito abajo del objetivo.
      errorAcumulado += error;  // Sumar el error actual al total

      // Limitar el error acumulado para que no se vaya al infinito
      if (errorAcumulado > 1000) errorAcumulado = 1000;
      if (errorAcumulado < -1000) errorAcumulado = -1000;

      float I = Ki * errorAcumulado;
      // Ki = 0.05, entonces si errorAcumulado es 100 → I = 5


      // --- COMPONENTE D (DERIVATIVO) ---
      // Mira qué tan RÁPIDO está cambiando el error.
      // Si el error está bajando muy rápido, significa que nos estamos
      // acercando MUY RÁPIDO al objetivo, ¡así que FRENA!
      // Es como frenar el auto ANTES de llegar al semáforo.
      float D = Kd * (error - errorAnterior);
      // Kd = 120.0, entonces si el error bajó 2°C desde la última vez → D = -240
      // Ese número negativo GRANDE hará que frenemos mucho

      errorAnterior = error;  // Guardar el error actual para la próxima vez


      // --- SUMAR LOS TRES COMPONENTES ---
      // La salida del PID es la suma de P + I + D
      float salidaPID = P + I + D;


      // --- CONVERTIR A PWM (0-255) ---
      // PWM es como un "control de potencia" para el calefactor.
      // 0 = apagado completamente
      // 255 = encendido a máxima potencia
      // 127 = encendido a "media potencia"
```

```
  potenciaPWM = (int)salidaPID;  // Convertir a número entero

  // Limitar entre 0 y 255
  if (potenciaPWM > 255) potenciaPWM = 255;
  if (potenciaPWM < 0) potenciaPWM = 0;


  // --- APLICAR LA POTENCIA AL CALEFACTOR ---
  analogWrite(mosfetPin, potenciaPWM);  // Escribir el valor PWM al MOSFET
                                        // analogWrite() controla la potencia


  // Decidir si consideramos al calefactor "encendido"
  // Si la potencia es mayor al 10% (25 de 255), está encendido
  calefactorEncendido = (potenciaPWM > 25);

} else {
  // Si el control está DESACTIVADO (botón OFF)...

  analogWrite(mosfetPin, 0);  // Apagar completamente el calefactor
  potenciaPWM = 0;
  calefactorEncendido = false;

  // Resetear las variables del PID
  errorAcumulado = 0;  // Borrar el error acumulado
  errorAnterior = 0;   // Borrar el error anterior
}


// --- PASO 3.3: DETECTAR SI CAMBIÓ LA VELOCIDAD ---
// Si el potenciómetro cambió más de 10 puntos, significa que
// el usuario está ajustando la velocidad, así que mostramos
// velocidad/dirección en el display por 3 segundos.

if (abs(valorPot - valorPotAnterior) > 10) {
  // abs() = valor absoluto (ignora si es positivo o negativo)
  // Pregunta: ¿el potenciómetro cambió más de 10 puntos?

  tiempoUltimoCambioMotor = millis();  // Marcar el momento del cambio
  valorPotAnterior = valorPot;         // Recordar el nuevo valor
}


// --- PASO 3.4: ACTUALIZAR EL DISPLAY ---
// Decidimos QUÉ mostrar en el display según si cambió la velocidad recientemente.

if (millis() - tiempoUltimoCambioMotor < 3000) {
  // Si pasaron MENOS de 3 segundos (3000 milisegundos) desde el último cambio...
  // MOSTRAR VELOCIDAD Y DIRECCIÓN

  lcd.setCursor(0, 0);  // Ir a la primera línea del display
  lcd.print("Velocidad: ");

  // Calcular el porcentaje de velocidad (0% a 100%)
  int velocidadPercent;
  if (motorApagado) {
    velocidadPercent = 0;  // Si el motor está apagado, mostrar 0%
  } else {
```

```
        // Convertir el delay del motor en porcentaje
        // AJUSTADO para microstepping 1/16:
        // delay grande (3000) = velocidad baja (0%)
        // delay pequeño (200) = velocidad alta (100%)
        velocidadPercent = map(delayMotor, 3000, 200, 0, 100);
        // Limitar entre 0 y 100 por seguridad
        if (velocidadPercent < 0) velocidadPercent = 0;
        if (velocidadPercent > 100) velocidadPercent = 100;
      }

      // Alinear los números (agregar espacios para que quede bonito)
      if (velocidadPercent < 100) lcd.print(" ");  // Si es menor a 100, agregar un
espacio
      if (velocidadPercent < 10) lcd.print(" ");   // Si es menor a 10, agregar otro
espacio

      lcd.print(velocidadPercent);  // Mostrar el número
      lcd.print("%    ");           // Mostrar el símbolo % y espacios para "limpiar"


      lcd.setCursor(0, 1);  // Ir a la segunda línea del display
      lcd.print("Dir: ");

      if (direccionActual == HIGH) {
        lcd.print("Adelante   ");  // Mostrar "Adelante" con espacios al final
      } else {
        lcd.print("Atras      ");  // Mostrar "Atras" con espacios al final
      }

    } else {
      // Si pasaron MÁS de 3 segundos...
      // MOSTRAR TEMPERATURA (modo normal)

      lcd.setCursor(0, 0);  // Ir a la primera línea
      lcd.print("T:");       // Mostrar "T:" (temperatura)
      lcd.print(temperaturaActual, 1);  // Mostrar temperatura con 1 decimal (ej: 23.5)
      lcd.print("C   ");     // Mostrar "C" (Celsius) y espacios

      // Mostrar el ESTADO del calefactor: [ON], [--], o [OFF]
      if (controlCalefactorActivo) {
        // Si el control está activado...
        if (calefactorEncendido) {
          lcd.print("[ON] ");  // Si está calentando, mostrar [ON]
        } else {
          lcd.print("[--]");   // Si está esperando (sin calentar), mostrar [--]
        }
      } else {
        // Si el control está desactivado...
        lcd.print("[OFF]");  // Mostrar [OFF]
      }


      lcd.setCursor(0, 1);  // Ir a la segunda línea
      lcd.print("Obj:");     // Mostrar "Obj:" (objetivo)
      lcd.print(tempObjetivo, 0);  // Mostrar temperatura objetivo sin decimales (ej: 240)
      lcd.print("C          ");    // Mostrar "C" y espacios para "limpiar"
    }
  }
```

```
}


// ==========================================
// ¡FIN DEL PROGRAMA!
// ==========================================
// El loop() se repetirá infinitamente mientras el Arduino esté encendido.
// El motor se moverá automáticamente gracias a la interrupción del Timer1.
// El PID controlará la temperatura de forma inteligente.
// ¡Y todo funcionará al mismo tiempo sin bloquearse! 🎉
```