

## CÓDIGO CONEXIÓN PC

```
/*
Este ejemplo permite apagar, encender o poner a parpadear el led interno del
Arduino.

Solicita por pantalla al usuario que índice la acción que desea realizar (apagar,
encender o
parpadear) y la envía por el puerto serie al Arduino.

El programa necesita que se le pase como parámetro el puerto serie a utilizar.
Por ejemplo:
    serie COM3
*/

#include <Windows.h>
#include <stdio.h>
#include <conio.h>
#include <locale.h>

/*
*/

int main(int argc, char* argv[])
{
    HANDLE hdPort;           // Manejador del puerto serie
    DCB paramsPort;          // Parámetros del puerto serie
    COMMTIMEOUTS tm;         // Tiempos de espera para el puerto serie
    char action;              // Acción a enviar al Arduino
    DWORD numBytes;          // Numero de bytes enviados al Arduino o
recibos del Arduino
    char serialName[20]; // Nombre del puerto serie
    char temperature[20]; // Temperatura enviada o recibida de Arduino. Se utiliza
una cadena de texto porque

    // hace más simple la programación aunque no sea lo más eficiente. Como máximo
la cadena puede tener 19 caracteres.

    // La cadena se reserva de 20 para poder contener los 19 caracteres máximos
más

    // el byte de fin de cadena. Ese byte no se envía o recibe del Arduino.
    float faux;              // Variable flotante auxiliar. Se utiliza
sólo para comprobar que la cadena "temperature" tiene un
                                // formato válido
    int res;                  // Resultado de algunas funciones

    // Se establece que el idioma es el español. Esto es sólo para que por la
// consola de Windows se representen bien los caracteres especiales del
// español (tildes, ñ, etc.)
    setlocale(LC_ALL, "spanish");

    // Se comprueba que se haya llamado al programa con un parámetro.
    // argc contiene el número de parámetros + 1, por eso se compara con 2

    if (argc != 2)
    {
        printf("Error: Número de parámetros incorrecto.\n");
        return -1;
    }

    // Los puertos serie en Windows se nombran como "COM1", "COM2", "COM3", etc.
```

```

    // Pero solo los nueve primeros se pueden abrir usando directamente ese
nombre.
    // A partir del 10, tiene que nombrarse como "\\.\COM10" "\\.\COM11" etc.
    // Por eso, al parámetro indicado por el usuario se le va a añadir la cadena
    // "\\." para completar el nombre.

    // Se almacena en serialName el nombre del puerto concatenando la cadena
    // inicial al nombre del puerto indicado por el usuario como parámetro.
    // Al utilizar sprintf_s se asegura que aunque el usuario introduzca un nombre
    // de puerto muy largo nunca se van a copiar más de 19 bytes y por tanto
    // no va desbordar el serialName.

    sprintf_s(serialName, 19, "\\.\COM%s", argv[1]);

    // Se abre el puerto indicando que se va a utilizar para lectura y escritura
    hdPort = CreateFile(serialName, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, 0, NULL);
    if (hdPort == INVALID_HANDLE_VALUE)
    {
        printf("Error: Imposible abrir el puerto serie\n");
        return 1;
    }

    // Una vez abierto hay que configurarlo para que su velocidad coincida con la
que se configura
    // en Arduino.

    // El puerto serie tiene bastantes parámetros de configuración. Para evitar
tener que indicar todos
    // lo más cómodo es leer la configuración que tiene actualmente, modificar
sólo lo que nos interesa
    // y volver a escribir la configuración

    // Se lee la configuración de los parámetros del puerto serie
    if (GetCommState(hdPort, &paramsPort) == 0)
    {
        printf("Error: Imposible configurar el puerto serie\n");
        return 2;
    }

    // Se modifican los parámetros necesarios para comunicar con Arduino

    paramsPort.BaudRate = CBR_9600;           // Velocidad: 9600 baudios
    paramsPort.Parity = NOPARITY;             // Sin paridad
    paramsPort.ByteSize = 8;                  // Numero de bits por byte 8
    paramsPort.StopBits = ONESTOPBIT;         // 1 bit de stop

    // Se vuelve a escribir la configuración

    if (SetCommState(hdPort, &paramsPort) == 0)
    {
        printf("Error: Imposible configurar el puerto serie\n");
        return 2;
    }

    // Ahora también hay que configurar los tiempos de espera por la respuesta del
Arduino. Se va
    // a configurar un tiempo de 500 ms. Es decir, cuando se va a leer por el
puerto serie, se espera
    // como máximo 500 ms por si llega algo.

    // El proceso de configuración es similar al anterior. Primero se leen los
valores actuales de los

```

// tiempos de espera, después se modifican los que hagan falta y por último se vuelven a escribir.

// Lectura de valores

```
if (GetCommTimeouts(hdPort, &tm) == 0)
{
    printf("Error: Imposible configurar el puerto serie\n");
    return 2;
}
```

// Se ajustan los valores

```
tm.ReadIntervalTimeout = 0;
tm.ReadTotalTimeoutMultiplier = 0;
tm.ReadTotalTimeoutConstant = 500;
```

// Se escriben los nuevos valores

```
if (SetCommTimeouts(hdPort, &tm) == 0)
{
    printf("Error: Imposible configurar el puerto serie\n");
    return 2;
}
```

// Bucle infinito de solicitud al usuario de la acción a realizar

do {

// Se le muestra al usuario las opciones que tiene

```
printf("\nIndique la acción a realizar\n");
printf("1 - Encender led\n");
printf("2 - Apagar led\n");
printf("3 - Forzar parpadeo led\n");
printf("4 - Leer temperatura\n");
printf("5 - Establecer temperatura deseada\n");
printf("0 - Salir\n");
printf("> ");
```

// Se lee la acción indicada por el usuario.

// \_getch lee el caracter introducido en la consola pero no lo escribe,

por eso

// después se hace el \_putch. De esta manera el usuario si ve en

pantalla la tecla

// pulsada

```
action = _getch();
_putchar(action);
```

// Según la acción...

```
switch (action)
```

```
{
```

```
case '0':
```

// Ha indicado salir. Se cierra el puerto y se finaliza

```
CloseHandle(hdPort);
```

```
return 0;
```

```
case '1':
```

```
case '2':
```

```
case '3':
```

// Para cualquiera de estas acciones se hace lo mismo, enviarla por el puerto serie al

// Arduino. Se escribe en el puerto, pasando como dirección del buffer a escribir directamente

// la dirección de la acción e indicando que sólo se va a escribir 1 byte.

```
if (WriteFile(hdPort, &action, 1, &numBytes, NULL) == 0)
{
```

```

        printf("\nError: Imposible enviar la acción.\n");
        return 4;
    }
    break;

    case '4':
        // Aquí también hay que enviar la acción al Arduino, pero
        // después hay que leer la respuesta
        // del Arduino con la temperatura
        if (WriteFile(hdPort, &action, 1, &numBytes, NULL) == 0)
        {
            printf("\nError: Imposible enviar la acción.\n");
            return 4;
        }

        // Se lee por el puerto la temperatura recibida. Se manda leer
        // bytes, porque tiene que quedar al menos uno libre para poner
        // como máximo 19
        // el final de la
        // cadena de texto
        if (ReadFile(hdPort, temperature, 19, &numBytes, NULL) == 0)
        {
            printf("\nError: Imposible leer la temperatura.\n");
            return 5;
        }

        // Se añade el final a la cadena de texto recibida. El final irá
        // en la posición
        // que corresponda a los bytes leídos. Por ejemplo, si se recibe
        // "12.35", el
        // valor de numBytes será 5 porque se han leído 5 caracteres y
        // la marca de final
        // de cadena habrá que ponerla en la posición de numBytes,
        // porque las cadenas en
        // C empiezan a contarse en 0. Es decir, quedará
        // temperature[0] = '1'
        // temperature[1] = '2'
        // temperature[2] = '.'
        // temperature[3] = '3'

        // temperature[4] = '5'
        // temperature[5] = '\0'
        temperature[numBytes] = '\0';

        // Se escribe el resultado en la pantalla
        printf("\nTemperatura: %s °C\n", temperature);
        break;

    case '5':
        // Se le pide al usuario la temperatura deseada. Se mete en un
        // bucle porque
        // se va a pedir de forma continua hasta que el usuario
        // proporcione un valor
        // de coma flotante válido.
        res = 0;
        while (res != 1)
        {
            printf("\nIntroduzca la temperatura deseada en °C: ");

            // Se recoge el valor en formato string
            res = scanf_s("%19s", temperature,
                (unsigned)_countof(temperature));
        }
    }
}

```

```

        if (res == 1)
        {
            // Antes de aceptar el string indicado por el
            usuario se comprueba que sea un          // valor en como flotante válido. Para ello
            utilizo la función sscanf_s.
            res = sscanf_s(temperature, "%f", &faux);
        }
        if (res != 1)
        {
            printf("Error: valor de temperatura inválido.
Introduzca un valor de coma flotante válido\n");
        }
    }

    // Se envía la acción al Arduino
    if (WriteFile(hdPort, &action, 1, &numBytes, NULL) == 0)
    {
        printf("\nError: Imposible enviar la acción.\n");
        return 4;
    }

    // Se envía el valor de temperatura deseado.
    if (WriteFile(hdPort, temperature, strlen(temperature),
&numBytes, NULL) == 0)
    {
        printf("\nError: Imposible enviar la consigna de
temperatura.\n");
        return 5;
    }

    break;
case EOF:
    printf("\nError: Fallo en la lectura desde consola\n");
    return 3;

default:
    printf("\nError: '%c' no es una acción válida\n", action);
}

} while (TRUE);

system("pause");
return 0;
}

```

A continuación, adjunto el correspondiente código en Arduino.

```

const int action = 0;           //Acción solicitada desde el PC
const int blinkLed = 0;        //Indicador de que el led interno debe
parpadear
const int ledLevel = LOW;      //Valor para el led interno
const int PIN_CALDERA = 2;     // Pin donde estará conectado el led que
indica el arranque de la caldera por temperatura más baja de la deseada

const float SCALE_FACTOR = 0.05; // Factor de escala para convertir el valor
leído del potenciómetro en grados centígrados.
float userTemperature = 0.0;    // Temperatura deseada por el usuario

void setup() {
    // Se abre el puerto serie con velocidad de 9600

```

```

Serial.begin(9600);

// Se espera de forma indefinida hasta que el puerto esté abierto
while (!Serial);

// Se configura el pin del led interno como salida
pinMode(LED_BUILTIN, OUTPUT);

// Se configura el pin 2 como salida.
pinMode(PIN_CALDERA, OUTPUT);
}

void loop() {
    float temperature;    // Temperatura en grados obtenida del potenciómetro
    // Se realiza el tratamiento de la temperatura
    temperature = control_temperature(userTemperature);

    // Se lee del puerto serie
    // La función retorna en action el primer byte que haya en el puerto
    // pendiente de leer y si no hay ninguno -1
    action = Serial.read();

    switch (action)
    {
        case '1': {
            // Se indica que el led debe encenderse y no parpadear
            ledLevel = HIGH;
            blinkLed = 0;
            break;
        }

        case '2': {
            // Se indica que el led debe apagarse y no parpadear
            ledLevel = LOW;
            blinkLed = 0;
            break;
        }

        case '3': {
            // Se indica que el led debe parpadear
            blinkLed = 1;
            break;
        }

        case '4': {
            // Se envía el valor de temperatura
            Serial.print(temperature);
            break;
        }

        case '5': {
            // Se lee el valor deseado para la temperatura transformándolo en
un float
            userTemperature = Serial.parseFloat();

            break;
        }

        // cualquier otra acción que se reciba, incluso que no se
reciba ninguna no altera el estado del led.
    }
}

```

```

    // Si hay que parpadear
    if (blinkLed)
    {
        // Se invierte la indicación para el led
        if (ledLevel == HIGH)
            ledLevel = LOW;
        else
            ledLevel = HIGH;

        // Se espera 500 ms
        delay(500);
    }

    // Se escribe el valor deseado en el led
    digitalWrite(LED_BUILTIN, ledLevel);
}

//-----
// control_temperature
// Lee el valor del potenciómetro que simula el sensor de temperatura del pin A0.
// Lo convierte a grados centígrados y compara ese valor con el valor deseado de
// temperatura indicado por el usuario. Si la temperatura es menor pone a 1 el
// pin
// donde esta conectado el arranque de la calefacción, sino lo pone a 0
//
// Necesita:
// - userGradTemp: Valor de temperatura deseada por el usuario
//
// Retorna: el valor de temperatura leído del potenciómetro
//-----
float control_temperature(float userGradTemp)
{
    int rawTemp = 0;    // Valor del potenciómetro
    float gradTemp = 0; // Temperatura en grados

    // Se supone que el potenciómetro que simula el sensor de temperatura
    // está conectado al pin A0

    // Se lee el valor del potenciómetro
    rawTemp = analogRead(A0);

    // Se transforma el valor leído a grados usando el factor de escala
    // indicado
    gradTemp = rawTemp * SCALE_FACTOR;

    // Si la temperatura es menor que la deseada, se enciende la caldera
    if (gradTemp < userGradTemp)
        digitalWrite(PIN_CALDERA, HIGH);
    else
        digitalWrite(PIN_CALDERA, LOW);

    // Se devuelve el valor
    return gradTemp;
}

```

## CÓDIGO COMPLETO ARDUINO

```
#include<LiquidCrystal.h>
#include<Keypad.h>
//declaro los pines a los que voy a conectar cada elemento
const int EchoPin = 10;
const int TriggerPin = 9;
const int PIN_CALDERA = 27;
#define LedPin 26
#define led_interior 28
#define pote A0
#define IN3 22
#define IN4 23
#define IN1 24
#define IN2 25
#define pulsador 2
#define pulsador_persiana_bajar 51
#define pulsador_persiana_subir 48
#define pulsador_luz 50
#define zumbador 8
#define ldr A1
#define DEBUG(a)

//declaro las variables que voy a utilizar en el código
int action = 0;
int blinkLed = 0;
int ledLevel = LOW;
//const int PIN_CALDERA = 2;
int dimension = 0;
const float SCALE_FACTOR = 0.05;
float userTemperature = 0.0;
long duration;
int distance, initialDistance, currentDistance, i;
int screenOffMsg = 0;
String password = "1234";
String tempPassword;
boolean activated = false; // estado de la alarma
boolean isActivated;
boolean activateAlarm = false;
boolean alarmActivated = false;
boolean enteredPassword; // estado de la contraseña para parar la alarma
boolean passChangeMode = false;
boolean passChanged = false;
const byte ROWS = 4; //4 filas
const byte COLS = 4; //4 columnas
char keypressed;
//defino los símbolos de las teclas del teclado
char keyMap[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = { 14, 15, 16, 17 }; //pines de las filas
byte colPins[COLS] = { 18, 19, 20, 21 }; //pines de las columnas
Keypad myKeypad = Keypad(makeKeymap(keyMap), rowPins, colPins, ROWS, COLS);
LiquidCrystal lcd(11, 12, 4, 5, 6, 7); // crea un objeto LCD. Parámetros: (rs,
enable, d4, d5, d6, d7). Indico donde conecto cada elemento del pin

void setup() {

  Serial.begin(9600);
```



```

    lcd.begin(16, 2); //lcd dividido en una matriz 16x2
    Serial.setTimeout(50);
    pinMode(pulsador_persiana_bajar, INPUT);
    pinMode(pulsador_persiana_subir, INPUT);
    pinMode(pulsador_luz, INPUT);
    pinMode(led_interior, OUTPUT);
    pinMode(LedPin, OUTPUT);
    pinMode(TriplePin, OUTPUT);
    pinMode(EchoPin, INPUT);
    pinMode(IN4, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(pote, INPUT);
    pinMode(pulsador, INPUT);
    pinMode(zumbador, OUTPUT);
    pinMode(ldr, INPUT);
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(PIN_CALDERA, OUTPUT);
    int distancia(int, int); //prototipos funciones
    void pausa(unsigned int);
    void termostato(int);

}

void loop() {

    float temperature;
    char *ppas;
    ppas = &password[0];
    //termostato(analogRead(pote));
    luz(digitalRead(ldr));
    persiana_subir(digitalRead(pulsador_persiana_subir));
    persiana_bajar(digitalRead(pulsador_persiana_bajar));
    //alarma(digitalRead(pulsador));
    timbre(digitalRead(pulsador));
    temperature = control_temperature(userTemperature);
    action = Serial.read();
    conexionpc(action, temperature);
    luz_interior(digitalRead(pulsador_luz));

    //Código Alarma
    if (activateAlarm) { //si se selecciona la opción de activar la alarma
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("La alarma se");
        lcd.setCursor(0, 1);
        lcd.print("activara en");

        int countdown = 9; // 9 segundos de cuenta atrás antes de activar
        la alarma
        while (countdown != 0) {
            lcd.setCursor(13, 1);
            lcd.print(countdown);
            countdown--;
            tone(zumbador, 700, 100);
            delay(1000);
        }
        lcd.clear();
        lcd.setCursor(0, 0);
    }
}

```

```

        lcd.print("Alarma Activada");
        initialDistance = distancia(TriggerPin, EchoPin); //toma la distancia
inicial medida por el ultrasonidos
        activateAlarm = false;
        alarmActivated = true;
    }
    if (alarmActivated == true) { //si la alarma esta activada
        currentDistance = distancia(TriggerPin, EchoPin) + 10; //se
comprueba la distancia actual con la inicial calculada antes
        if (currentDistance < initialDistance) { //si la distancia es menor
de la inicial
            tone(zumbador, 1000); // Se activa el zumbador
            lcd.clear();
            enterPassword(); //se llama a la función para introducir la
contraseña
        }
    }
    if (!alarmActivated) { //si la alarma no está activada
        if (screenOffMsg == 0) {
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print("A - Activar");
            lcd.setCursor(0, 1);
            lcd.print("B - Cambiar contraseña");
            screenOffMsg = 1;
        }
        keypressed = myKeypad.getKey();
        if (keypressed == 'A') { //Si se elige A se activa la alarma
            tone(zumbador, 1000, 200);
            activateAlarm = true;
        }
        else if (keypressed == 'B') { //si se elige la B se pide primero la
contraseña actual
            lcd.clear();
            int i = 1;
            tone(zumbador, 2000, 100);
            tempPassword = "";
            lcd.setCursor(0, 0);
            lcd.print("Contrasena:");
            lcd.setCursor(0, 1);
            lcd.print(">");
            passChangeMode = true;
            passChanged = true;
            while (passChanged) {
                keypressed = myKeypad.getKey();
                if (keypressed != NO_KEY) {
                    if (keypressed == '0' || keypressed == '1' ||
keypressed == '2' || keypressed == '3' ||
                    keypressed == '4' || keypressed == '5'
|| keypressed == '6' || keypressed == '7' ||
                    keypressed == '8' || keypressed == '9')
{
                        tempPassword += keypressed;
                        lcd.setCursor(i, 1);
                        lcd.print("*");
                        i++;
                        tone(zumbador, 2000, 100);
                    }
                }
                if (i > 5 || keypressed == '#') { //Si excedemos el
número de dígitos o pulsamos # nos la vuelve a pedir
                    tempPassword = "";

```

```

        i = 1;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Contraseña:");
        lcd.setCursor(0, 1);
        lcd.print(">");
    }
    if (keypressed == '*') {
        i = 1;
        tone(zumbador, 2000, 100);
        if (password == tempPassword) { //si la
contraseña es correcta se pide la nueva contraseña
            tempPassword = "";
            lcd.clear();
            lcd.setCursor(0, 0);
            //-----
            lcd.print("Digitos:"); //nos pide el
número de dígitos que queremos en la nueva contraseña
            lcd.setCursor(0, 1);
            lcd.print(">");
            while (passChangeMode) {
                keypressed = myKeypad.getKey();
                if (keypressed != NO_KEY) {
                    if (keypressed == '0' ||
keypressed == '1' || keypressed == '2' || keypressed == '3' ||keypressed == '4'
|| keypressed == '5' || keypressed == '6' || keypressed == '7' ||
keypressed == '8' || keypressed == '9')
{
                    dimension += keypressed;
                    lcd.setCursor(i, 1);
                    lcd.print("*");
                    i++;
                    tone(zumbador, 2000, 100);
                }

                lcd.clear();
                ppas = (char*)realloc(ppas, dimension * sizeof(char));
                //-----
                lcd.print("Nueva contraseña:");
                lcd.setCursor(0, 1);
                lcd.print(">");
                while (passChangeMode) {
                    keypressed = myKeypad.getKey();
                    if (keypressed != NO_KEY) {

                        if (keypressed == '0' || keypressed == '1' || keypressed == '2' ||
keypressed == '3' ||keypressed == '4' || keypressed == '5' || keypressed
== '6' || keypressed == '7' ||keypressed == '8' || keypressed == '9')
                        {

                            tempPassword += keypressed;

                            lcd.setCursor(i, 1);

                            lcd.print("*");
                            i++;

                            tone(zumbador, 2000, 100);
                        }
                    }
                }
            }
        }
    }
}

```

```

if (i > dimension + 1 || keypressed == '#') { //se vuelve a pedir la nueva contraseña

    tempPassword = "";

    i = 1;

    tone(zumbador, 2000, 100);

    lcd.clear();

    lcd.setCursor(0, 0);

    lcd.print("Nueva contrase a");

    lcd.setCursor(0, 1);

    lcd.print(">");

}

if (keypressed == '*') { //se guarda la nueva contrase a
i = 1;
tone(zumbador, 2000, 100);
password = tempPassword;
passChangeMode = false;
passChanged = false;
screenOffMsg = 0;

}

}

}

}

}

}

}

}

}

//-----
void enterPassword() { //funcion para introducir la contrase a por teclado
int k = 5;
tempPassword = "";
activated = true;
lcd.clear();//limpiamos pantalla
lcd.setCursor(0, 0);//cursor en la parte superior izq de la pantalla
lcd.print(" *** ALARMA *** ");
lcd.setCursor(0, 1);//cursor en la parte inferior izq de la pantalla
lcd.print("Cont>");
while (activated) {
    keypressed = myKeypad.getKey();
    if (keypressed != NO_KEY) {
        if (keypressed == '0' || keypressed == '1' || keypressed == '2' || keypressed == '3' || keypressed == '4' || keypressed == '5' || keypressed == '6' || keypressed == '7' || keypressed == '8' || keypressed == '9') {
            tempPassword += keypressed;
            lcd.setCursor(k, 1);
            lcd.print("*");
            k++;
        }
    }
    if (k > dimension + 1 || keypressed == '#') { //si la contrase a supera los d gitos o se presiona # se vuelve a pedir la contrase a

```

```

        tempPassword = "";
        k = dimension;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(" *** ALARMA *** ");
        lcd.setCursor(0, 1);
        lcd.print("Cont>");
    }
    if (keypressed == '*') { //si se presiona * entonces se comprueba
si la contraseña introducida es correcta
        if (tempPassword == password) {
            activated = false;
            alarmActivated = false;
            noTone(zumbador);
            screenOffMsg = 0;
        }
        else if (tempPassword != password) {
            lcd.clear();
            lcd.setCursor(0, 1);
            lcd.print("Error");
            delay(2000);
            lcd.clear();
            lcd.setCursor(0, 0);
            lcd.print(" *** ALARMA *** ");
            lcd.setCursor(0, 1);
            lcd.print("Cont>");
        }
    }
}

//-----
int distancia(int TriggerPin, int EchoPin) //funcion distancia del ultrasonidos
{
    long duration, distance;
    digitalWrite(TriggerPin, LOW);
    delayMicroseconds(4);
    digitalWrite(TriggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(TriggerPin, LOW);

    duration = pulseIn(EchoPin, HIGH); //calculo el tiempo
    distance = duration * 10 / 292 / 2; // calculo la distancia en cm
    return distance;
}

//-----
void luz(int valor)//enciende la luz en funcion de la lectura del ldr
{
    //Serial.println(valor);
    if (valor == HIGH)
    {
        digitalWrite(LedPin, LOW);
    }
    else
    {
        digitalWrite(LedPin, HIGH);
    }
}

//-----

```

```

void luz_interior(int valor)//función que enciende la luz en función de la
lectura del pulsador
{
    if (valor == LOW)
    {
        digitalWrite(led_interior, HIGH);
    }
    else
    {
        digitalWrite(led_interior, LOW);
    }
}
//-----
void persiana_subir(int valorPulsador)//función que sube la persiana si el valor
del pulsador es 1
{
    //Serial.println(valorPulsador);//Mirar los pines según sentido del motor
    if (valorPulsador == LOW)
    {
        digitalWrite(IN3, LOW);
        digitalWrite(IN4, HIGH);
    }
    else
    {
        digitalWrite(IN4, LOW);
    }
}
//-----
void persiana_bajar(int valorPulsador)//función que baja la persiana si el valor
del pulsador es 1
{
    if (valorPulsador == LOW)
    {
        digitalWrite(IN3, HIGH);
        digitalWrite(IN4, LOW);
    }
    else
    {
        digitalWrite(IN3, LOW);
    }
}
//-----
void alarma(int valorPulsador)//
{
    int cm = distancia(TriggerPin, EchoPin);
    if (valorPulsador == LOW)
    {
        if (cm <= 19)
        {
            analogWrite(zumbador, 300);
            pausa(5000);
            analogWrite(zumbador, LOW);
        }
    }
    else
    {
        analogWrite(zumbador, LOW);
    }
}

```

```

//-----
void timbre(int valorPulsador)//función que activa el timbre al pulsar el
pulsador
{
    if (valorPulsador == HIGH)
    {
        analogWrite(zumbador, 884);
    }
    else
    {
        digitalWrite(zumbador, LOW);
    }
}
//-----

void pausa(unsigned int milisegundos)//función para crear una pausa sin usar el
delay
{
    volatile unsigned long compara = 0;
    volatile int contador = 0;
    do
    {
        if (compara != millis())
        {
            contador++;
            compara = millis();
        }
    } while (contador <= milisegundos);
    return;
}
//-----
// control_temperature
// Lee el valor del potenciómetro que simula el sensor de temperatura del pin A0.
// Lo convierte a grados centígrados y compara ese valor con el valor deseado de
// temperatura indicado por el usuario. Si la temperatura es menor pone a 1 el
pin
// donde está conectado el arranque de la calefacción, sino lo pone a 0
//
// Necesita:
// - userGradTemp: Valor de temperatura deseada por el usuario
//
// Retorna: el valor de temperatura leído del potenciómetro
//-----
float control_temperature(float userGradTemp)
{
    int rawTemp = 0;    // Valor del potenciómetro
    float gradTemp = 0; // Temperatura en grados

    // Se supone que el potenciómetro que simula el sensor de temperatura
    // está conectado al pin A0

    // Se lee el valor del potenciómetro
    rawTemp = analogRead(A0);

    // Se transforma el valor leído a grados usando el factor de escala
    // indicado
    gradTemp = rawTemp * SCALE_FACTOR;
}

```

```

    // Si la temperatura es menor que la deseada, se enciende la caldera
    if (gradTemp < userGradTemp)
        digitalWrite(PIN_CALDERA, HIGH);
    else
        digitalWrite(PIN_CALDERA, LOW);

    // Se devuelve el valor
    return gradTemp;
}
//-----
void conexionpc(int action, float temperature)//función que conecta el puerto
serie al Arduino
{
    switch (action)
    {
        case '1': {
            // Se indica que el led debe encenderse y no parpadear
            ledLevel = HIGH;
            blinkLed = 0;
            break;
        }

        case '2': {
            // Se indica que el led debe apagarse y no parpadear
            ledLevel = LOW;
            blinkLed = 0;
            break;
        }

        case '3': {
            // Se indica que el led debe parpadear
            blinkLed = 1;
            break;
        }

        case '4': {
            // Se envia el valor de temperatura
            Serial.print(temperature);
            break;
        }

        case '5': {
            // Se lee el valor deseado para la temperatura transformándolo en
un float
            userTemperature = Serial.parseFloat();

            break;
        }
    }
    if (blinkLed)
    {
        // Se invierte la indicación para el led
        if (ledLevel == HIGH)
            ledLevel = LOW;
        else
            ledLevel = HIGH;

        // Se esperan 500 ms
        delay(500);
    }
}

```



```
    // Se escribe el valor deseado en el led  
    digitalWrite(LED_BUILTIN, ledLevel);  
}
```