Funcionamiento General del programa

El programa cumple con los siguientes procesos:

- Detecta movimiento con un sensor PIR.
- Si la alarma está activada, y hay movimiento, activa un buzzer intermitente y muestra un mensaje de alerta en la pantalla LCD.
- Si no hay movimiento por 5 segundos, actualiza la pantalla para mostrar "sin movimiento".
- Muestra en el LCD si la alarma está activada, desactivada o si hay movimiento.
- Se puede activar o desactivar la alarma con un botón físico.

A continuación se presenta un esquema general del funcionamiento del programa:

```
+---------------------+
|      setup()        |  → Inicializa hardware y LCD
+---------------------+
          |
          v
+---------------------+
|       loop()        |  → Bucle principal del programa
|---------------------|
| Leer botón          |
| Cambiar estado de   |
| la alarma           |
|---------------------|
| Leer sensor PIR     |
|---------------------|
| ¿Alarma activa?     |  → Si la alarma está activada y hay movimiento
|        |            |
|        v            |
|   ¿Movimiento?      |
|     /     \         |
|  Sí       No        |
|  /         \        |
| Buzzer    LCD msg   |
|---------------------|
| ¿Tiempo sin mov.?   |  → Si no hay movimiento por 5 segundos, la alarma se
desactiva
|        |            |
|        v            |
|  LCD "sin mov."     |
+---------------------+
```

Describción del código del programa

Librerías utilizadas

```
#include <Arduino.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Arduino.h

Librería estándar de Arduino para el manejo de pines digitales y analógicos, incluye funciones para leer y escribir en pines, así como para manejar temporizadores. Permite utilizar instrucciones tales como:

- *setup()*, *loop()*
- *digitalRead()*, *digitalWrite()*
- *pinMode()*
- *millis()*, *delay()*, etc.

Wire.h

Librería que permite la comunicación I2C entre el Arduino y dispositivos externos, de tal manera que se puede complementar nuestro circuito con sensores, pantallas LCD, RTC, expansores de pines, etc. Permite utilizar comandos como:

- *pinMode(pin, INPUT)* - Configura un pin especificado como entrada digital para leer señales externas.
- *pinMode(pin, OUTPUT)* - Configura un pin especificado como salida digital para enviar señales.
- *digitalWrite(pin, HIGH)* - Envía un nivel alto (voltaje) un pin configurado como salida.
- *digitalWrite(pin, LOW)* - Envía un nivel bajo (voltaje) un pin configurado como salida.
- *int estado = digitalRead(pin)* - Lee el valor digital (HIGH o LOW) desde un pin configurado como entrada.
- *int valor = analogRead(pin)* - Lee el valor analógico (de 0 a 1023) de un pin analógico.
- *analogWrite(pin, valor)* - Envía una señal PWM un pin para simular niveles analógicos (valor de 0 a 255).

LiquidCrystal_I2C.h

Una librería que permite controlar pantallas LCD usando un adaptador I2C (muy común en módulos LCD 16x2 o 20x4). Permite utilizar comandos como:

- *lcd.init()* — inicia la pantalla
- *lcd.setCursor(col, fila)* — posiciona el cursor
- *lcd.print("texto")* — escribe texto en pantalla
- *lcd.clear()* — borra el contenido
- *lcd.backlight()* — enciende la luz de fondo

Variables y objetos utilizados

Los números 2, 8, y 4 en estas líneas:

```
const int pinPIR = 2;
const int pinBuzzer = 8;
```

```
const int pinBoton = 4;
```

representan números de pines físicos del Arduino

¿Qué significan?

Pines de la Protoboard

```
    const int pinPIR = 2;
```

- El sensor de movimiento PIR está conectado al pin digital 2 del Arduino.
- Este pin se configura como entrada, porque el sensor envía una señal que el Arduino debe leer.

```
   const int pinBuzzer = 8;
```

- El buzzer (alarma) está conectado al pin digital 8.
- Este pin se configura como salida, porque el Arduino le envía corriente para hacer sonar el buzzer.

```
   const int pinBoton = 4;
```

- El botón físico está conectado al pin digital 4.
- Este también es un pin de entrada, ya que el Arduino debe detectar si el botón se presionó.

Se usa const int para:

- Asignar un nombre legible a cada pin (más fácil que recordar "¿qué era el pin 8?").
- No modificar por accidente más adelante en el programa (es constante).

Puedes conectar los componentes a otros pines digitales si modificas los valores en el código. Por ejemplo, si conectas el buzzer al pin 7 en lugar del 8, tendrías que cambiar:

```
const int pinBuzzer = 7;
```

Siempre y cuando no elijas:

- Pines digitales (no todos los pines del Arduino tienen funciones iguales).
- El pin no estén siendo usados por otro componente.

Pantalla LCD del Arduino

```
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Crea un objeto llamado lcd que representa una pantalla LCD con interfaz I2C, este será el nombre del objeto que usarás para controlar la pantalla (como lcd.print(...), lcd.setCursor(...), etc.).

```
(0x27, 16, 2)
```

Son los parámetros de configuración:

Parámetro	Significado
`0x27`	Dirección I2C de la pantalla (puede ser `0x27`, `0x3F`, etc.).
`16`	Número de columnas (caracteres por línea) del LCD.
`2`	Número de filas del LCD.

"Voy a usar una pantalla LCD con interfaz I2C que tiene 16 columnas y 2 filas. Su dirección I2C es 0x27. Llamaré a esta pantalla lcd para controlarla desde el código."

Variables de control de la alarma

```
bool alarmaActiva = true;
```

Indica si la alarma está activada o no y por defecto estará activada. Cada vez que se pulsa el boton , se activa o desactiva la alarma.

```
bool movimientoDetectado = false;
```

Indica si ha habido movimiento detectado o no y por defecto no habrá movimiento detectado. Esta variable ha de actualizarse constantemente en la función loop().

```
bool lcdNecesitaActualizar = true;
```

Se usa para evitar actualizar la pantalla LCD innecesariamente, de tal manera que en la pantalla se actualiza el mensaje cuando algo cambia (como desactivar la alarma o terminar el movimiento).

Variables del botón

```
int valorBotonActual = HIGH;
```

Guarda el estado actual del botón (presionado o no).

```
int valorBotonAnterior = HIGH;
```

Guarda el estado anterior del botón, de tal manera que esta variable nos permitira saber si el boton fue presionado o no al compararse con el valor de la variable valorBotonActual.

```
unsigned long tiempoUltimoCambio = 0;
```

Guarda el momento en que cambió por última vez el estado del botón. La variable ha de ser usada para hacer el "debounce" (evitar falsos positivos por rebote mecánico del botón).

```
const unsigned long debounceDelay = 50;
```

Tiempo mínimo (en milisegundos) para aceptar un cambio de estado del botón como válido, en este caso está configurado en 50. *Podemos pensarlo como el tiempo mínimo que debe pasar antes de el sistema pueda detectar presión en el botón de nuevo*

Variables del buzzer (sonido intermitente)

```
unsigned long tiempoUltimaActivacionBuzzer = 0;
```

Guarda cuándo fue la última vez que se activó o desactivó el buzzer, se usa para que el buzzer suene de forma intermitente (ON-OFF-ON...).

```
const long intervaloBuzzer = 250;
```

Intervalo en milisegundos entre cada cambio de estado del buzzer (cada 250 ms), lo que produce un efecto de parpadeo sonoro cuando hay movimiento.

Variables de detección sin movimiento

```
unsigned long tiempoUltimoMovimiento = 0;
```

Guarda el tiempo en que se detectó el último movimiento, ha de ser utilizado para saber si ha pasado un tiempo determinado sin movimiento.

```
const unsigned long tiempoEsperaSinMovimiento = 5000;
```

Tiempo de espera (en milisegundos) para considerar que ya no hay movimiento. En este caso, después de 5 segundos sin detección, el LCD cambia el mensaje a “Sin movimiento”.

Tabla de resumen de variables y constantes

Variable	¿Para qué sirve?
`alarmaActiva`	Saber si la alarma está encendida o no
`movimientoDetectado`	Saber si el sensor PIR detectó algo
`lcdNecesitaActualizar`	Controlar si se debe refrescar la pantalla
`valorBotonActual/Anterior`	Detectar cambios en el botón
`tiempoUltimoCambio`	Saber cuándo fue el último cambio del botón
`debounceDelay`	Filtro para ignorar rebotes del botón
`tiempoUltimaActivacionBuzzer`	Saber cuándo se cambió el estado del buzzer
`intervaloBuzzer`	Controlar con qué frecuencia suena el buzzer
`tiempoUltimoMovimiento`	Saber cuándo se detectó movimiento por última vez
`tiempoEsperaSinMovimiento`	Tiempo antes de mostrar "Sin movimiento"