

Sistema de Alarma con Sensor PIR y Notificación LCD

Integrantes:

- Muñoz Santillán, Angelo David
 - Murillo Gálvez, Lenin Jahir
 - Nogales Zapata, Sebastián Alejandro
 - Oña Maigua, Maylis Emilie
 - Oña Pinenla, Anderson Francisco
-

1. Introducción

Este documento establece un conjunto de reglas y buenas prácticas para la escritura de código en el entorno de Arduino, aplicadas al desarrollo de un sistema de alarma. El objetivo es asegurar que el proyecto sea legible, mantenible y eficiente. Para ello, se toma como base el código del sistema de alarma, analizando su estructura, formato, convenciones y las técnicas de programación utilizadas.

2. Nomenclatura de Variables y Funciones

Para mantener la claridad, se utilizará un sistema de nomenclatura consistente:

- **Variables:** Se usará la convención **camelCase**, comenzando con minúscula. Se añaden prefijos descriptivos para indicar el propósito de la variable.
 - pin...: para pines de hardware (ej: pinPIR, pinBuzzer).
 - tiempo...: para variables que almacenan marcas de tiempo con millis() (ej: tiempoInicioArmado).
 - duracion...: para intervalos de tiempo constantes (ej: duracionRetardoArmado).
 - estado...: para variables que guardan el estado actual (ej: estadoActualAlarma).
 - contador...: para variables que llevan la cuenta de eventos (ej: contadorDisparosAlarma).
- **Constantes:** Se escribirán en **UPPER_SNAKE_CASE** (mayúsculas y guiones bajos). Esto aplica a valores fijos que no cambiarán.
 - Ej: `const int MAX_PULSOS_BUZZER = 10;`
- **Funciones:** Se nombrarán usando **camelCase**, comenzando con un verbo que describa la acción que realizan.

- Ej: verificarConexionPIR(), actualizarPantallaLCD().
- Las funciones estándar de Arduino (setup, loop) mantienen su formato en minúsculas.
- **Enumeraciones (enum):** El tipo de la enumeración se nombra en **PascalCase** (ej: EstadoAlarma) y sus miembros en **UPPER_SNAKE_CASE** (ej: ALARMA_ACTIVADA, ALARMA_DISPARRADA).

3. Estilo de Comentarios

Los comentarios son cruciales para entender el código. Se usarán de la siguiente manera:

- **Comentarios de Bloque:** Para separar y describir secciones principales del código (Definición de Pines, Variables de Estado, etc.).

```
C++  
  
// --- Definición de Pines ---  
const int pinPIR = 2; // Pin donde está conectado el sensor PIR
```

- **Comentarios de Cabecera de Función:** Cada función debe tener un comentario que explique qué hace, qué parámetros recibe y qué retorna.

```
C++  
  
// --- Función para verificar la conexión del PIR ---  
// Retorna true si el PIR está conectado y funcional, false en caso contrario.  
bool verificarConexionPIR() {  
    // ...  
}
```

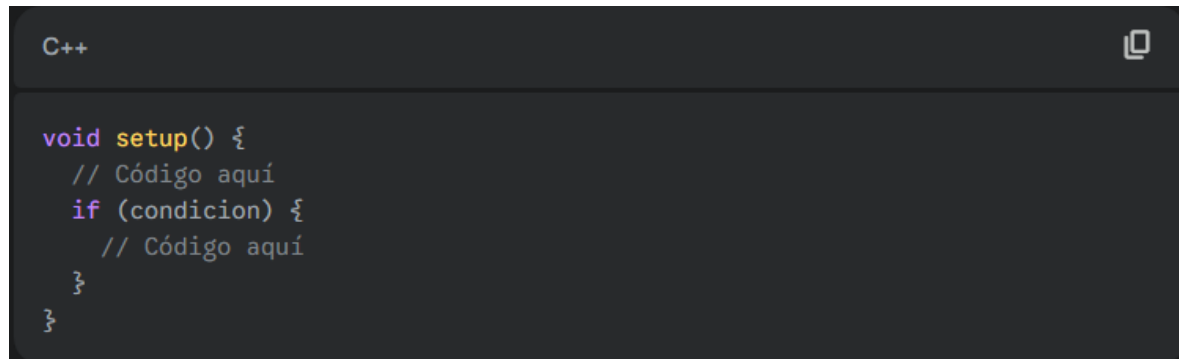
- **Comentarios de Línea:** Para aclarar líneas de código específicas cuya función no sea inmediatamente obvia.

```
C++  
  
lcd.init();      // Inicializa el LCD  
lcd.backlight(); // Enciende la luz de fondo del LCD
```

4. Formato del Código

Una organización visual limpia es fundamental para la legibilidad.

- **Indentación:** Se usarán **2 espacios** para cada nivel de indentación.
- **Llaves {}:** La llave de apertura se colocará **en la misma línea** que la declaración de la estructura de control (if, for, switch). La llave de cierre irá en su propia línea.



```
C++  
  
void setup() {  
    // Código aquí  
    if (condición) {  
        // Código aquí  
    }  
}
```

- **Espaciado:** Se dejará un espacio alrededor de los operadores (=, +, ==, etc.) y después de las comas. Se usarán líneas en blanco para separar bloques lógicos.

5. Orden y Estructura del Código

El archivo .ino se organizará siempre en el siguiente orden:

1. **Inclusión de Librerías (#include)**
2. **Definición de Constantes y Pines (const int)**
3. **Configuración de Objetos Globales (LiquidCrystal_I2C lcd(...))**
4. **Declaración de Variables Globales**
5. **Definición de Funciones Personalizadas (verificarConexionPIR())**
6. **Función setup()**
7. **Función loop()**

6. Buenas Prácticas a Seguir

- **No bloquear el loop():** Evitar el uso de delay() en el bucle principal. Usar millis() para gestionar el tiempo sin detener la ejecución.

- **Máquina de Estados:** Utilizar una máquina de estados con enum y switch-case para lógicas complejas, como se ve en el ejemplo (EstadoAlarma).
 - **Usar const:** Declarar como const cualquier variable que no deba cambiar su valor (pines, duraciones).
 - **Antirrebote (Debounce):** Implementar un mecanismo de antirrebote para entradas como botones y evitar lecturas múltiples.
 - **Nombres Descriptivos:** Usar nombres claros y autoexplicativos para variables y funciones.
 - **Modularización:** Dividir el código en funciones pequeñas con un propósito único.
 - **Feedback para Depuración:** Usar Serial.print() durante el desarrollo para monitorear el comportamiento del sistema.
-

7. Ejemplos: Cómo Hacerlo vs. Cómo No Hacerlo

Gestión del Tiempo

✅ **Correcto (No bloqueante con millis()):**

```
C++  
  
// Bueno  
unsigned long tiempoAnterior = 0;  
const long intervalo = 1000;  
  
void loop() {  
    unsigned long tiempoActual = millis();  
    if (tiempoActual - tiempoAnterior >= intervalo) {  
        tiempoAnterior = tiempoActual;  
        // ... cambiar estado del LED ...  
    }  
    // ... leer estado del botón aquí, siempre funciona ...  
}
```

✗ Incorrecto (Bloqueante con delay()):

C++



```
// Malo
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000); // El código se detiene aquí por 1 segundo
    digitalWrite(ledPin, LOW);
    delay(1000); // Y aquí también. No se puede leer un botón.
}
```

Manejo de Estados

✓ Correcto (Máquina de estados con enum):

C++



```
// Bueno
enum EstadoSemaforo { VERDE, AMARILLO, ROJO };
EstadoSemaforo estadoActual = ROJO;

void loop() {
    switch (estadoActual) {
        case VERDE:
            // ... lógica para el estado VERDE ...
            break;
        case AMARILLO:
            // ... lógica para el estado AMARILLO ...
            break;
        case ROJO:
            // ... lógica para el estado ROJO ...
            break;
    }
}
```

✗ Incorrecto (Múltiples if con "números mágicos"):

C++

```
// Malo
int estado = 0; // 0=rojo, 1=verde, 2=amarillo. ¿Qué significan?

void loop() {
  if (estado == 1) {
    // ...
  } else if (estado == 2) {
    // ...
  } else if (estado == 0) {
    // ...
  }
  // ¿Y si añadimos un estado de "intermitente"? Se vuelve un lío.
}
```