Control programado con Arduino

Práctica 9: Modularización con Funciones (Paso de Parámetros)

A medida que nuestros proyectos crecen en tamaño y complejidad, escribir todo el código de corrido dentro de la función `loop` hace que sea difícil de leer, depurar y mantener. En esta práctica daremos el salto a la programación estructural. Aprenderás a dividir tu programa en bloques independientes llamados **funciones**, diseñando tu propio patrón de parpadeo personalizado mediante el paso de parámetros físicos (pin de salida y tiempo de retardo).

Objetivos de Aprendizaje

Al finalizar esta sesión, serás capaz de:

- Comprender el concepto de modularización y la estructura de una función en C++.
- Declarar y definir funciones personalizadas con tipo de retorno vacío (`void`).
- Implementar el paso de parámetros (`int` , `int`) para flexibilizar el comportamiento de las funciones.
- Llamar a funciones personalizadas desde el bucle principal `loop` de forma limpia y organizada.

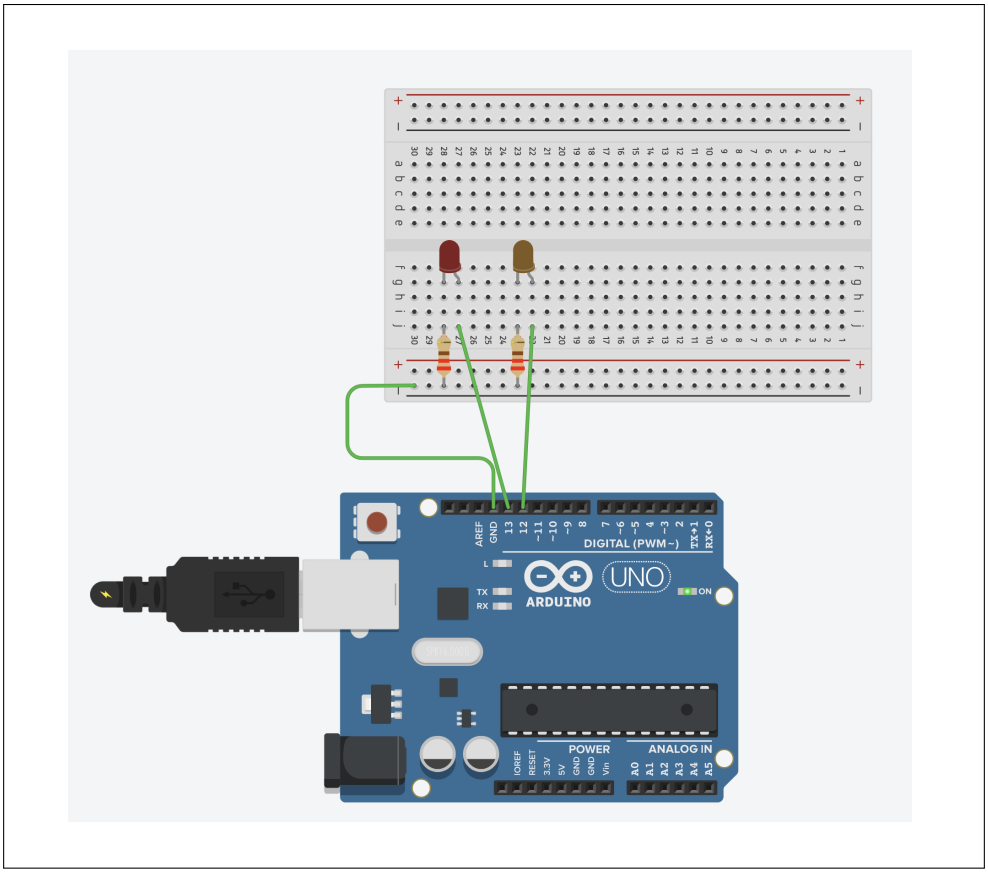
Componentes Necesarios

Para simplificar la parte física de esta práctica y centrarnos de lleno en el aprendizaje del código, recuperaremos y ampliaremos el circuito con dos diodos de salida de la Práctica 2.

- 1 Placa Arduino Uno.
- 1 Placa de pruebas pequeña (Protoboard).
- 2 Diodos LED (te sugerimos usar 1 Verde y 1 Rojo).
- 2 Resistencias de $220\,\Omega$ (una para proteger cada LED).
- Cables de conexión virtuales.

Esquema de Montaje en la Protoboard

Asociaremos cada color a un pin digital secuencial del Arduino para facilitar el direccionamiento por software.



Código Base: Declaración, Parámetros y Llamada

Escribe el siguiente programa en tu editor de TinkerCad. Observa que ahora la función `loop` es extremadamente corta y fácil de leer, ya que el trabajo pesado se ha delegado a una nueva función llamada .

```
// Practica 9: Modularizacion de un patron de parpadeo con funciones
int ledVerde = 13;
int ledRojo = 12;

void setup() {
  pinMode(ledVerde, OUTPUT);
  pinMode(ledRojo, OUTPUT);
}

void loop() {
  // Llamamos a nuestra funcion personalizada indicando que pin y que tiempo
    usar
  parpadearLed(ledVerde, 1000); // Parpadea el led verde cada 1 segundo
  parpadearLed(ledRojo, 500);   // Parpadea el led rojo cada 500 milisegundos
  parpadearLed(ledVerde, 200);  // Parpadea el led verde muy rapido (200 ms)
}

// 1. DEFINICION DE LA FUNCION PERSONALIZADA
// 'void' significa que la funcion no devuelve ningun valor al terminar.
// Entre parentesis definimos dos variables locales (parametros de entrada).
void parpadearLed(int pinLed, int tiempoRetardo) {
  digitalWrite(pinLed, HIGH);  // Enciende el LED indicado por el parametro
  delay(tiempoRetardo);        // Espera el tiempo indicado por el parametro
  digitalWrite(pinLed, LOW);   // Apaga el LED indicado
  delay(tiempoRetardo);        // Espera el tiempo indicado
}
```

Listing 1: Código utilizando una función personalizada con paso de parámetros.

¿Cómo funciona el código?

Una función es un subprograma que agrupa un conjunto de instrucciones bajo un nombre único para realizar una tarea específica.

- **Estructura de la cabecera**: `void` `int` `int`
  - `void`: Es el tipo de datos que devuelve la función. En este caso no devuelve nada, solo ejecuta acciones, por lo que usamos la palabra clave `void` (vacío).
  - : Es el nombre identificador que elegimos para nuestra función.
  - y : Son los **parámetros de entrada**. Actúan como variables locales dentro de la función. Reciben sus valores en el mismo instante en el que llamamos a la función desde el bucle principal.
- **La llamada a la función**: Cuando el programa lee , se detiene momentáneamente en la función `loop` , copia el valor de (13) dentro de y el valor 1000 dentro de , y salta a ejecutar las instrucciones de la función personalizada. Al terminar, regresa de inmediato a la siguiente línea de `loop` .

El Reto de la Práctica 9

Pon a prueba tus dotes de diseño de software modular superando las siguientes propuestas de programación:

1. **Función de Encendido Simultáneo**: Diseña una nueva función personalizada llamada `encenderAmbos(int pinA, int pinB, int tiempo)` que encienda los dos LEDs a la vez, espere el tiempo indicado por parámetro, los apague a la vez, y vuelva a esperar ese mismo tiempo. Pruébala llamándola desde tu `loop()`.
2. **Función de Código Morse**: Diseña una función especializada llamada `emitirSOS(int pinLed)` que, al recibir un pin de salida, reproduzca en ese LED la secuencia clásica de socorro en código Morse (S.O.S): 3 parpadeos muy cortos, 3 parpadeos largos y 3 parpadeos muy cortos.
3. *Pregunta para reflexionar*: Si en lugar de usar una función personalizada hubieras querido hacer parpadear los dos LEDs con tres ritmos de tiempo diferentes (como en el `loop()` base), ¿cuántas líneas de código habrías tenido que escribir de forma repetitiva? ¿Qué ventajas aporta modularizar en términos de limpieza y prevención de errores tipográficos?