Programación para Sistemas Embebidos

4. Estructuras de control y Funciones

Vamos a estudiar ahora las llamadas "Estructuras de Control" en programación. Estas estructuras son herramientas esenciales que nos permiten tomar decisiones y repetir acciones en función de ciertas condiciones. Son como las instrucciones que seguimos en la vida cotidiana, como "si hace calor, entonces usa una remera" o "haz ejercicio durante 30 minutos todos los días".

Estructuras de control condicionales

La estructura condicional más común es el "if" ("si", en español). Con esta estructura, podemos evaluar una condición y ejecutar un bloque de código solo si esa condición es verdadera. Veamos un ejemplo:

```
int temperatura = 25;
if (temperatura > 30) {
  // Si la temperatura es mayor que 30 grados, ejecutamos este bloque de código
  Serial.println("Hace mucho calor. Enciende el ventilador.");
} else {
  // Si la temperatura no es mayor que 30 grados, ejecutamos este bloque de código
  Serial.println("El clima es agradable. Disfruta tu día.");
}
```

En este ejemplo, comparamos la variable "temperatura" con el valor 30. Si la temperatura es mayor que 30, se imprime el mensaje "Hace mucho calor. Enciende el ventilador.". De lo contrario, se imprime "El clima es agradable. Disfruta tu día.".

***Nota**: Uso de la función `Serial.print()` en Arduino*

La función `Serial.print()` es una de las herramientas más valiosas en la programación de Arduino. Permite enviar datos desde el microcontrolador a la interfaz de comunicación serial, como el monitor serial de la IDE de Arduino o una conexión UART a través de USB. Esta función es extremadamente útil para la depuración y el monitoreo en tiempo real de los valores y estados de nuestro programa.

Sintaxis: Serial.print(dato);

La función `Serial.print()` acepta varios tipos de datos, incluyendo números enteros, decimales, caracteres y cadenas de texto.

La función `Serial.println()` es similar e incluye un salto de linea en la consola, por lo que las podemos combinar para obtener un formateo básico de la información mostrada.

Importancia y utilidad:

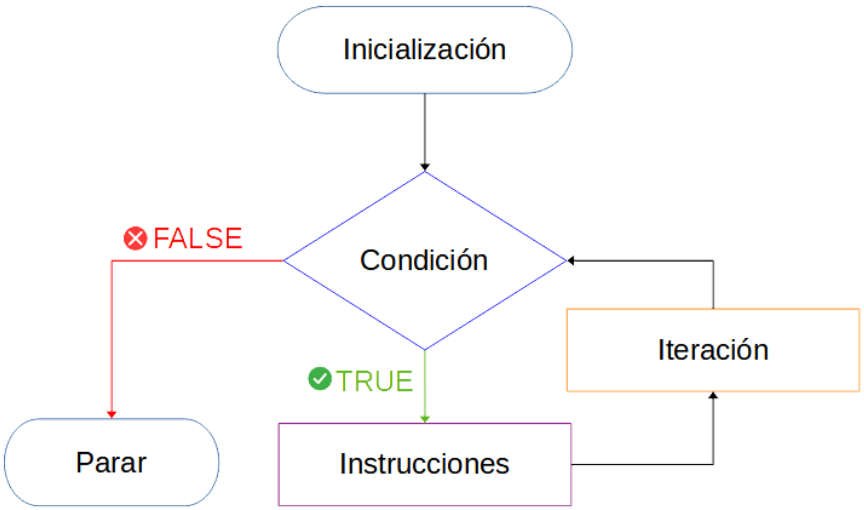
- Facilita el seguimiento del comportamiento del programa: Al utilizar `Serial.print()`, podemos verificar si los datos que estamos utilizando son los esperados y realizar un seguimiento del flujo de ejecución del programa.

- Depuración de errores: Cuando surgen problemas en nuestro código, podemos imprimir los valores de las variables en puntos clave del programa para entender mejor lo que está ocurriendo y encontrar posibles errores.

- Prueba y verificación: Al mostrar información relevante durante la ejecución del programa, podemos confirmar que los sensores están proporcionando los valores correctos o si las operaciones matemáticas se realizan adecuadamente.

Estructuras de repetición

Las estructuras de bucles nos permiten repetir acciones varias veces.



Dentro de las estructuras de repetición el bucle "for" es uno de los más utilizados en sistemas embebidos. Veamos cómo funciona:

```
for (int i = 0; i < 5; i++) {
  // Este bloque de código se ejecutará 5 veces, con i tomando los valores de 0 a 4.
  Serial.println("Hola, esto es un saludo número " + String(i));
}
```

En este ejemplo, creamos un bucle "for" que se ejecutará 5 veces. La variable "i" se inicializa en 0 y aumenta de uno en uno con cada iteración. Dentro del bucle, imprimimos un mensaje de saludo, utilizando el valor de "i" para contar cuántas veces se ha ejecutado el bucle.

La instrucción Mientras (while)

Otra estructura de repetición es la instrucción while. Esta estructura nos permite repetir un bloque de código mientras se cumpla una condición especificada.

La instrucción `while` es especialmente útil cuando queremos realizar una tarea repetidamente hasta que se cumpla una cierta condición de salida.

Ejemplo utiliza la instrucción `while` para leer un sensor analógico y encender un LED en función del valor leído:

```
const int sensorPin = A0;   // Pin del sensor analógico conectado al pin A0
const int ledPin = 13;      // Pin del LED conectado al pin 13
void setup() {
  pinMode(ledPin, OUTPUT);  // Establecer el pin del LED como salida
  Serial.begin(9600);       // Iniciar la comunicación serial para depuración
}
void loop() {
  digitalWrite(ledPin, LOW);    // Apagar el LED
  int valorSensor = analogRead(sensorPin);  // Leer el valor del sensor analógico
  // Mientras el valor del sensor sea mayor que 500, encender el LED, de lo contrario, apagarlo.
  while (valorSensor > 500) {
        digitalWrite(ledPin, HIGH);   // Encender el LED
        Serial.println("LED Encendido");
        // Esperar un breve período para evitar una repetición muy rápida del bucle
        delay(500);
        digitalWrite(ledPin, LOW);    // Apagar el LED
        Serial.println("LED Apagado");
        // Leer nuevamente el valor del sensor para actualizar la condición del bucle
        valorSensor = analogRead(sensorPin);
  }
}
```

En este ejemplo, utilizamos la instrucción `while` para controlar el encendido y apagado del LED en función del valor del sensor analógico conectado al pin A0. Mientras el valor del sensor sea mayor que 500, el LED permanecerá encendido y apagado con un breve retraso entre cada cambio.

Es importante tener en cuenta que la condición del bucle `while` se evalúa antes de cada iteración. Por lo tanto, si el valor del sensor se mantiene por encima de 500, el LED continuará encendiéndose y apagándose en un bucle mientras el programa se ejecuta en el bucle principal (`loop()`).

¿Cuándo usar la instrucción `while`?

La instrucción `while` es útil cuando queremos repetir un bloque de código en función de una condición específica. Por ejemplo, podemos usarla para leer una entrada analógica hasta que alcance cierto valor, o para mantener una acción en bucle hasta que se presione un botón.

Advertencia: Es importante tener precaución al usar la instrucción `while` para evitar bucles infinitos que puedan bloquear la ejecución del programa. Asegúrense de que la condición de salida sea alcanzable en algún momento, o utilicen otras estructuras de control como `for` si desean una iteración controlada.Por otra parte, en un programa para Arduino, ya tenemos un bucle infinito construido con loop() el cual es de uso obligatorio para el compilador, muchas veces, usaremos este bucle para recorrer las condiciones de nuestro programa y no un bucle while infinito como es necesario en otros lenguajes.

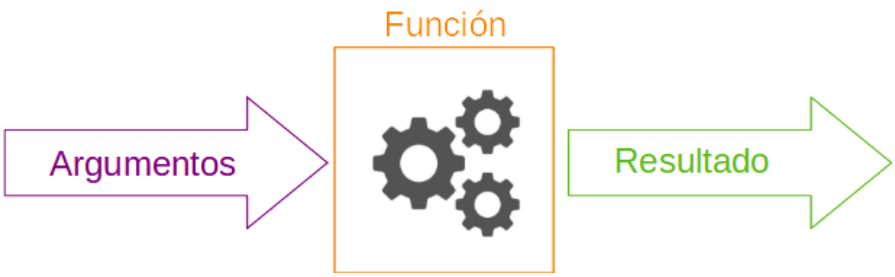
Importancia de las estructuras de control

Las estructuras de control son fundamentales porque nos permiten escribir programas más inteligentes y eficientes. Podemos tomar decisiones y realizar acciones basadas en datos variables, y también podemos repetir tareas automáticamente, ahorrando tiempo y esfuerzo.

En Arduino, utilizamos estas estructuras para controlar actuadores, como LEDs y motores, dependiendo de sensores o datos de entrada. También podemos crear patrones y secuencias repetitivas para hacer que nuestros proyectos sean más interactivos y dinámicos. Son herramientas clave que todo programador debe conocer y dominar.

Funciones

Las funciones son fundamentales en la programación porque nos permiten dividir un programa en tareas más pequeñas y manejables. Esto hace que el código sea más claro, más organizado y más fácil de mantener. Además, el uso de funciones nos permite reutilizar el código, lo que ahorra tiempo y esfuerzo al evitar repetir el mismo código en diferentes partes del programa.



Una función es un bloque de código definido por un nombre y que puede ser reutilizado o ejecutado desde diferentes puntos en un programa en C. La función se crea para realizar una tarea específica y es una forma efectiva de organizar y reutilizar el código en un programa. Al igual que las variables, el nombre de una función debe ser único dentro del programa. Además, una vez definida una función, puede ser utilizada desde cualquier lugar del programa siempre y cuando la función esté dentro del alcance adecuado.

Ejemplo de función para Arduino:

Supongamos que queremos crear una función para encender y apagar un LED conectado al pin 13 de Arduino. Podemos crear una función llamada "encenderApagarLED" que realice esta tarea.

```
// Declaración de la función encenderApagarLED
void encenderApagarLED() {
  digitalWrite(13, HIGH); // Encender el LED
  delay(1000); // Esperar 1 segundo (1000 milisegundos)
  digitalWrite(13, LOW); // Apagar el LED
  delay(1000); // Esperar 1 segundo nuevamente
}
void setup() {
  pinMode(13, OUTPUT); // Configurar el pin 13 como salida para el LED
}
void loop() {
  // Llamar a la función encenderApagarLED
  encenderApagarLED();
}
```

Explicación del ejemplo:

En el ejemplo anterior, hemos definido una función llamada `encenderApagarLED` que realiza la tarea de encender y apagar un LED conectado al pin 13 de Arduino. La función está declarada antes del `setup` y `loop`, lo que la hace global y, por lo tanto, puede ser llamada desde cualquier lugar del programa.

Dentro del `setup`, configuramos el pin 13 como una salida para el LED, y en el `loop`, llamamos a la función `encenderApagarLED`. Cada vez que la función se llama desde el `loop`, el LED se enciende durante un segundo y luego se apaga durante otro segundo, creando un efecto de parpadeo.

Creando prototipos de funciones

Un prototipo de función es una declaración anticipada de una función en un programa antes de que se defina su implementación. En otras palabras, es una forma de indicar al compilador cómo se verá y qué tipo de datos aceptará la función antes de que se utilice o se defina completamente en el programa.

El prototipo de función incluye información sobre el nombre de la función, el tipo de dato que devuelve (si es una función que devuelve un valor) y los tipos de datos de los parámetros que acepta la función. Esto le permite al compilador saber cómo manejar las llamadas a la función y verificar si los argumentos proporcionados coinciden con los esperados por la función.

La declaración del prototipo de función se coloca generalmente al comienzo del archivo de código o en un archivo de encabezado (header file) para que esté disponible en todo el programa, antes de que se utilice la función en cualquier parte del código. Esto es especialmente útil cuando se tienen funciones que se llaman entre sí o cuando se desea separar la definición de la función (la implementación) de su uso en diferentes partes del programa.

Tomemos un ejemplo sencillo, queremos crear una función que sume dos enteros y devuelva el resultado.

Hay dos argumentos que son variables de tipo entero. En este caso, el resultado de la suma de estos dos valores enteros también es un valor entero. No tiene que serlo, pero para este ejemplo lo es. El prototipo en ese caso sería:

```
        int sumador(int m, int n);
```

Cuerpo e instrucciones de las funciones

Como probablemente has entendido intuitivamente, el cuerpo de una función es el lugar donde ocurre todo; es donde se construyen todas las instrucciones de la función.

Imagina el cuerpo de la función como un bloque real, puro y nuevo de código fuente. Aquí puedes declarar y definir variables, agregar condiciones y utilizar bucles. También, como ya introdujimos antes, puedes devolver el valor de una variable.

Echemos un vistazo al cuerpo de nuestro ejemplo de función sumador:

```
int sumador(int m, int n) // esta línea es la cabecera de la función
{
  int result; // esta es la variable para almacenar el resultado
  result = m + n; // realiza la suma y almacena el resultado en 'result'
  return result; // devuelve el valor de la variable 'result'
}
```

En la línea `int result;`, declaramos una variable llamada 'result' y su alcance es el mismo que el de los argumentos. En `result = m + n;`, se realizan dos operaciones, y ya sabes que el operador '+' tiene una precedencia mayor que el operador '=', lo cual es excelente, ya que primero se realiza la operación matemática y luego se almacena el resultado en la variable 'result'.

Aquí es donde ocurre la magia; tomamos dos operandos y los combinamos en uno solo. Recuerda que, en una combinación de múltiples operaciones matemáticas, debemos tener en cuenta el orden de precedencia; es crítico para evitar resultados inesperados.

Por último debemos llamar a la función desde el bucle principal:

```
// Declaración de función personalizada para sumar dos enteros
int sumador(int a, int b);
void setup() {
  // Iniciar la comunicación serie a 9600 baudios
  Serial.begin(9600);
  Serial.println("vamos a sumar todos los enteros de 2 en 2 menores que 100, e imprimirlos por consola");
}
void loop() {
  // Variable para almacenar el resultado de la suma
  int suma;

  // Bucle para sumar los enteros desde 0 hasta 99 de dos en dos
  for (int i = 0; i < 100; i += 2) {
    // Llamar a la función personalizada y almacenar el resultado en currentResult
    suma = sumador(i, i + 1);

    // Mostrar el resultado en el monitor serie
    Serial.println(suma);
  }

  // Hacer una pausa de 2 segundos
  Serial.println("Espera");
  delay(2000);
}
// Definición de la función personalizada para sumar dos enteros
int sumador(int a, int b) {
  int result; // esta es la variable para almacenar el resultado
  result = m + n; // realiza la suma y almacena el resultado en 'result'
  return result; // devuelve el valor de la variable 'result'
}
```