Naan Mudhalvan

Industrial IoT and Industry 4.0

Tektork Private Limited, Coimbatore

Arduino Coding

Arduino Coding

- Arduino coding is a simplified form of C++ used to program Arduino boards.
- It's built on a specific structure that makes it easy for beginners to control electronics.
- The core of any Arduino sketch (the term for an Arduino program) consists of two main functions:
  - void setup() and void loop().

The Core Structure

Every Arduino sketch must have these two functions:

void setup():

This function runs only once when the Arduino board is powered on or reset. It's used for initialization tasks, such as setting up pin modes (e.g., as an input or output) and starting serial communication. Think of it as the preparation phase for your program.

void loop():

This function runs continuously after void setup() is complete. It's the main part of your code where you perform actions repeatedly, like reading sensor data, turning on an LED, or responding to button presses. This continuous repetition is what makes Arduino ideal for controlling real-world devices.

Key Concepts

- **Pins**: The Arduino board has multiple **pins** that serve as connections to external components like LEDs, sensors, and motors. These pins can be digital (on or off) or analog (a range of values).
- **Functions**: Just like in C++, you can create your own functions to organize your code and reuse blocks of instructions.
- **Variables**: You use **variables** to store data, such as sensor readings or a counter for a loop.
- **Libraries**: **Libraries** are collections of pre-written code that simplify complex tasks. For example, a motor library can handle the intricate details of controlling a motor, so you don't have to write that code yourself.
- **Comments**: You can add comments to your code using // for single-line comments or /* */ for multi-line comments. These are essential for explaining what your code does.

- **Basic Structure**

```
// Global Variable & Library Section
// Place libraries, constants, and global variables here

void setup()
{
  // This runs once at the start
  // Initialize variables, pin modes, libraries, Serial communication, etc.
}

void loop()
{
  // This runs repeatedly after setup()
  // Core logic of the program goes here
}
```

Void Setup()

- A void setup() is a function used in the Arduino programming language.
- It is the first function called when an Arduino board is powered on or reset.
- Its primary purpose is to initialize settings and variables that will be used throughout the program.

Key Characteristics of void setup()

- **Execution**: The code inside the void setup() function runs only **once**.

- **Initialization**: It's where you typically set up the **initial state** of your project.

This includes:

- **Pin Modes**: Configuring the digital pins on the Arduino board as either an **input** (to read sensor data) or an **output** (to control components like LEDs or motors). You do this using the pinMode() function.

- **Serial Communication**: Starting communication with a computer via the Serial Monitor to display data or debug your code, using the Serial.begin() function.

- **Libraries**: Initializing and configuring any external libraries you are using, such as those for controlling an LCD screen or a Wi-Fi module.

Common Uses of setup()

Pin configuration

- `pinMode(LED_BUILTIN, OUTPUT);  // Set built-in LED pin as output`
- `pinMode(2, INPUT);             // Set pin 2 as input`

Start Serial Communication

- `Serial.begin(9600);  // Start serial monitor at 9600 baud rate`

Initialize sensors, modules, or libraries

- `lcd.begin(16, 2);   // Initialize 16x2 LCD`
- `WiFi.begin("SSID", "PASSWORD");  // Start WiFi connection`

Set default states

- `digitalWrite(LED_BUILTIN, LOW); // Ensure LED is OFF initially`

Void Loop()

- A void loop() is a function in the Arduino programming language that is a fundamental part of every sketch.
- It's where the majority of your program's logic resides and, as the name suggests, it runs continuously.

How void loop() Works

- After the void setup() function runs once at the start of your program, the void loop() function begins and executes its code repeatedly, forever.
- This continuous execution makes it perfect for tasks that need to be constantly checked or updated, such as:

  - **Reading sensor data:** Checking the temperature, light level, or distance from an object.
  - **Controlling actuators:** Turning a motor on or off, or adjusting the brightness of an LED.
  - **Monitoring inputs:** Checking if a button has been pressed.

How void loop() Works

- After the void setup() function runs once at the start of your program, the void loop() function begins and executes its code repeatedly, forever.
- This continuous execution makes it perfect for tasks that need to be constantly checked or updated, such as:

  - **Reading sensor data:** Checking the temperature, light level, or distance from an object.
  - **Controlling actuators:** Turning a motor on or off, or adjusting the brightness of an LED.
  - **Monitoring inputs:** Checking if a button has been pressed.

Common Uses of loop()

Blink an LED continuously

```
void loop()
{
  digitalWrite(13, HIGH);   // LED ON
  delay(1000);              // wait 1 second
  digitalWrite(13, LOW);    // LED OFF
  delay(1000);              // wait 1 second
}
```

Read sensor values repeatedly

```
void loop()
{
  int sensorValue = analogRead(A0); // Read analog pin A0
  Serial.println(sensorValue);      // Print value to Serial Monitor
  delay(500);                       // Small delay
}
```

Control actuators based on conditions

```
void loop()
{
  int button = digitalRead(2);    // Read button on pin 2
  if (button == HIGH)
{
    digitalWrite(13, HIGH);       // Turn LED ON
  }
else
{
    digitalWrite(13, LOW);        // Turn LED OFF
  }
}
```

- **Sections of an Arduino Sketch**

a) Global Declaration Section

- Import libraries (#include <library.h>)
- Define constants (#define LED_PIN 13)
- Declare global variables

```
#include <Wire.h>
#define LED_PIN 13
int counter = 0;
```

b) Void setup() Function

- Executed only **once** when the Arduino starts or resets.
- Used to:
  - Set **pin modes** (pinMode())
  - Start serial communication (Serial.begin())
  - Initialize sensors/actuators

```
void setup()
{
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}
```

c) Void loop() Function

- Runs **continuously** in a cycle after setup() finishes.
- Place the main logic here (reading sensors, controlling actuators, etc.)

```
void blinkLED(int times)
{
  for(int i=0; i<times; i++)
{
    digitalWrite(LED_PIN, HIGH);
    delay(300);
    digitalWrite(LED_PIN, LOW);
    delay(300);
  }
}
```

Common Functions

Function	Purpose
pinMode()	Set pin as INPUT or OUTPUT
digitalWrite()	Write HIGH or LOW to a pin
digitalRead()	Read digital value from a pin
analogRead()	Read analog value (0–1023)
analogWrite()	Output PWM signal
delay()	Pause execution for milliseconds

pinMode()

- The pinMode() function is used to configure a specific digital pin as either an **input** or an **output**.
- This is a crucial first step in any program that uses digital pins, and it's typically called once in the setup() function.

pinMode(pin, mode):

- pin: The number of the digital pin you want to configure.
- mode: The mode for the pin, which can be **INPUT**, **OUTPUT**, or **INPUT_PULLUP**. When a pin is set to INPUT_PULLUP, it activates an internal pull-up resistor, which is useful for stabilizing the state of a pin when a button or switch is not pressed.
- Example
  - `pinMode(2, INPUT); // Set pin 2 as input (e.g., button)`
  - `pinMode(13, OUTPUT); // Set pin 13 as output (e.g., LED)`

Digital I/O function in Arduino

digitalWrite()

- The digitalWrite() function sets a digital pin to a specific voltage level: **HIGH** or **LOW**. This is used to control output devices like LEDs, relays, or motors.

Syntax: digitalWrite(pin, value):

- pin: The number of the pin you want to write to.
- value: The value to set the pin to, which can be **HIGH** (approximately 5V or 3.3V, depending on the board) or **LOW** (0V).
- Example
  - `digitalWrite(13, HIGH); // Turn LED on`
  - `digitalWrite(13, LOW); // Turn LED off`

digitalRead()

- The digitalRead() function reads the voltage level of a digital pin and returns a value of either **HIGH** or **LOW**.
- This is used to read the state of input devices like buttons or sensors.

Syntax:digitalRead(pin):

- pin: The number of the pin you want to read from.
- The function returns **HIGH** if the pin is high (close to the board's operating voltage) and **LOW** if the pin is low (close to 0V).
- Example
  - `int buttonState = digitalRead(2); // Read state of button on pin 2`

Analogl I/O function in Arduino

analogWrite()

- The analogWrite() function writes an analog value to a digital pin. On most Arduino boards, this is done using **Pulse Width Modulation (PWM)**, a technique that simulates an analog voltage by rapidly switching a digital pin between HIGH and LOW.
- This is useful for controlling the brightness of an LED, the speed of a motor, or the position of a servo.

Syntax: analogWrite(pin, value)

- pin: The number of the digital pin you want to write to. **Not all digital pins support PWM**.
- On an Arduino Uno, these pins are marked with a tilde (~) symbol (e.g., ~3, ~5, ~6, ~9, ~10, ~11).

Analogl I/O function in Arduino Contd

- value: The duty cycle of the PWM signal, which is an integer from **0 to 255**. A value of 0 is equivalent to 0V (always OFF), and a value of 255 is equivalent to the maximum voltage (always ON).
- A value in between, such as 127, produces a signal that is on about 50% of the time.
- **Example**
  - `analogWrite(ledPin, ledBrightness);// Write the mapped value to the LED pin to control its brightness`

analogRead()

- The analogRead() function reads the voltage from a specified analog pin and converts it into a digital value.
- This is typically used for reading data from analog sensors like potentiometers, photoresistors, or temperature sensors.

Syntax: analogRead(pin)

- pin: The number of the analog input pin you want to read from (e.g., A0, A1, etc.).
- **Return Value**: The function returns an integer from **0 to 1023**. This 10-bit resolution corresponds to the voltage range of the microcontroller (usually 0V to 5V or 0V to 3.3V).
- A value of 0 corresponds to 0V, and a value of 1023 corresponds to the maximum voltage (e.g., 5V).
- Example
- `int potValue = analogRead(potPin);// Read the value from the potentiometer (0-1023)`

delay()

- The delay() function in Arduino is a command that pauses the program for a specified duration.
- It's used to create timed intervals, allowing you to control the timing of events in your code.

Syntax: delay(ms)

- ms: The number of milliseconds to pause the program. A millisecond is one-thousandth of a second.
- `delay(1000);// Wait for 1000 milliseconds (1 second)`

- **Blocking Function**:
  - delay() is a **blocking** function, meaning it halts all other code execution until the specified time has passed.
  - While the delay() function is running, your Arduino cannot read sensor data, respond to button presses, or perform any other tasks. This behavior is a significant limitation for more complex projects.

millis()

- The millis() function in Arduino is a non-blocking alternative to delay().
- It's a function that returns the number of milliseconds that have passed since the Arduino board began running the current program.

Syntax: millis()

Return Value:

- It returns an **unsigned long** integer representing the number of milliseconds since the program started.
- An unsigned long is used because the count can become quite large, and it will "roll over" back to zero after approximately 50 days.
- Examples
  - `unsigned long previousMillis = 0; Declaration`

This is where the main logic for non-blocking timing resides.

- unsigned long currentMillis = millis();: Get the current time in milliseconds.
- if (currentMillis - previousMillis >= interval): This is the core of the non-blocking timing. It checks if the time that has passed (currentMillis - previousMillis) is greater than or equal to the desired interval.
- previousMillis = currentMillis;: If the interval has passed, the program updates the previousMillis to the current time, resetting the timer.

Thank You