Typedef Struct

Kennedy Freitas

In C and C++ programming, a
`struct` (short for **structure**) is used to group related variables together under a single name. The `typedef struct` pattern simplifies how we refer to these structures. Let's go step by step:

1️⃣ What is `typedef struct`?

- `typedef struct` creates a **custom data type** that holds multiple related variables.
- Instead of managing multiple separate variables, you **organize them neatly** inside a structure.
- This improves **code readability, maintainability, and scalability**.

How `typedef struct` is Used in The Code?

The code has **three structures**:

🟢 (1) LED Configuration Struct

```
typedef struct {
  int red1;
  int yellow1;
  int green1;
  int red2;
  int green2;
} LEDConfig;
```

📌 What does this do?

- Defines a structure named `LEDConfig` that groups **all LED pins**.

- Instead of managing five separate `int` variables, you **store them in one structured group**.

USAGE:

```
LEDConfig leds = {5, 6, 9, 10, 11};
```

- This means:
  - `leds.red1 = 5`
  - `leds.yellow1 = 6`
  - `leds.green1 = 9`
  - `leds.red2 = 10`
  - `leds.green2 = 11`
- Now, whenever you need to access an LED, you **refer to it as** `leds.red1`, `leds.green1`, etc.

🟢 (2) Sensor Configuration Struct

```
typedef struct {
  int trigger;
  int echo;
} SensorConfig;
```

📌 What does this do?

- Groups the **HC-SR04 ultrasonic sensor pins** (Trigger and Echo).
- Instead of managing two separate variables, they are **organized inside** `SensorConfig`.

Usage

```
SensorConfig sensor = {12, 13};
```

- This means:
  - `sensor.trigger = 12`
  - `sensor.echo = 13`
- Now, when referencing the sensor, you use `sensor.trigger` and `sensor.echo`, making it clear **which sensor the variables belong to**.

🟢 (3) Timing Configuration Struct

```
typedef struct {
  unsigned long previousMillis;
  const unsigned long green2Duration;
  const unsigned long sensorReenable;
} TimingConfig;
```

📌 What does this do?

- Groups **timing-related values** for LED delays and sensor reactivation.
- `previousMillis` stores the **last recorded time** for event tracking.
- `green2Duration` sets the **delay before Ledgreen2 turns ON**.
- `sensorReenable` sets the **delay before movement detection resets**.

Usage

```
TimingConfig timing = {0, 10000, 15000};
```

- This means:

- `timing.previousMillis = 0` (initial time tracking)
- `timing.green2Duration = 10000` (Ledgreen2 turns on again after **10 seconds**)
- `timing.sensorReenable = 15000` (Sensor resets after **15 seconds**)

- Whenever you need to reference a delay, just use:

```
timing.green2Duration
timing.sensorReenable
```

5️⃣ Summary

Feature	Without `typedef struct`	With `typedef struct`
Variable Organization	Many separate variables	Groups related variables together
Readability	Hard to track individual variables	Clear, logical grouping
Code Maintenance	Updating is tedious	Change values easily in one place
Error Reduction	Possible mix-ups in pin numbers	Clear variable associations

---

🔷 Final Thoughts

By using `typedef struct`, your code is now:
✅
More readable

✅ **Easier to maintain**

✅ **Less error-prone**

Now, when adding **new sensors, LEDs, or timing settings**, you **just modify the struct** instead of rewriting multiple variables.