Perfect! Let's go **week by week**, starting with **Phase 0 – Mindset Reset**, and I'll cover **everything end-to-end** so you know exactly what to do before touching a single line of C. This is the foundation that separates "someone who writes C code" from "someone who engineers embedded systems."

Phase 0: Mindset Reset (Week 0)

Goal: Build the right mental model for embedded engineering. This week is about **thinking like firmware**, not coding yet.

Day 1–2: Understand the Hardware Context

You must understand what an LRU (Line Replaceable Unit) is, how it communicates, and why firmware matters.

Tasks:

1. **Read and Understand LRU Architecture**
    - Each card is a "mini-computer":
        - CPU / MCU
        - Memory (RAM, ROM, Flash)
        - Communication ports (SPI, UART, I2C, Ethernet)
        - Power management
        - Built-In Test (BIT) circuitry
    - How cards talk to each other over backplane
    - Understand what happens when one card fails
2. **Ask Yourself:**
    - How does software control hardware?
    - How do you read a status register?
    - How do you trigger a hardware action safely?

Deliverable: Draw a block diagram of your LRU system. Include:

- Power flow
- Communication links
- CPU / memory / peripherals

Day 3–4: Binary, Bits, and Registers

Embedded engineers manipulate **hardware at the bit level**. This is non-negotiable.

Topics to Learn:

1. Binary and hexadecimal number systems
2. Bitwise operations:

    - AND (`&`), OR (`|`), XOR (`^`), NOT (`~`)
    - Shift left (`<<`) and shift right (`>>`)
3. Masks, flags, and bitfields
4. Endianness (big vs little endian)
5. Signed vs unsigned numbers

Mini Exercises:

- Convert decimal to binary/hex
- Create masks to set/clear specific bits
- Extract a bitfield from a register value
- Predict what happens after a left/right shift

Example Problem:

> Register `0xABCD1234`, mask `0xFF00`, extract the second byte.

Day 5–6: Memory Map Understanding

Your firmware talks directly to memory-mapped registers.

Learn:

- What a memory map is
- Difference between:
    - Flash (ROM)
    - SRAM
    - Memory-mapped peripherals
- How addresses relate to hardware functions

Exercise:

- Take a sample datasheet (UART, GPIO, Timer)
- Identify:
    - Base address
    - Offset for each register
    - Bit definitions

Deliverable: Table showing register addresses and their functions for one peripheral (like GPIO).

Day 7: Embedded Thinking Mindset

Key principle:

> Embedded programming is **predictable control over hardware**. Not apps. Not GUIs. Not "running code in a PC".

Concepts to internalize:

1. **Determinism** – Firmware must always behave predictably.
2. **Fail-safety** – LRU must fail gracefully, never catastrophically.
3. **Timing** – You're dealing with clocks, delays, interrupts.
4. **Registers as Interfaces** – Each peripheral exposes functionality as memory addresses.
5. **Testing is coding** – Firmware isn't done until it's verified on hardware.

Mini Task: Write a 1-page summary on:

- How your LRU talks to hardware
- What can go wrong if your code is wrong
- How timing, bits, and registers affect reliability

End of Phase 0 Deliverables (Week 0)

By the end of this week, you should have:

1. **LRU block diagram** – showing all cards, power, and communication
2. **Bit manipulation exercises** – working in paper or simple Python if you want
3. **Memory map table** – at least one peripheral fully mapped
4. **Embedded mindset summary** – your notes on determinism, fail-safety, timing, and registers

---

✅ **Phase 0 Outcome:**
After this week, you're no longer a "C language beginner." You are now thinking like an embedded engineer. You understand **hardware first**, so when we start C next week, it's just translating hardware control into code.

If you want, I can **lay out Phase 1 (Week 1–3) next**, which is where we **start writing C**, but only with hardware-focused exercises like registers, pointers, and memory mapping — no fluff.

Do you want me to continue with that next?