Assignments ::

🛠️ Assignment 1: ATM Machine with Limited Cash (RAII + Memory Control)

- **Topics covered**: Ownership, Borrowing, RAII, Stack/Heap usage.
- **Task**: Simulate an ATM machine:
    - Cash is stored as a struct with total amount.
    - Each withdrawal attempts to move/borrow from the cash store.
    - Ensure memory safety and no double free using ownership tracking.
    - Cash is automatically released (RAII) when ATM shuts down.
- **Challenge**: Print memory address & segment info (stack/heap) during operations for awareness.

🏦 Assignment 2: Loan Approval System (Option + Result Based Workflow)

- **Topics covered**: Option, Result, Custom Error Types, Pattern Guards.
- **Task**:
    - Take inputs: income, age, loan amount.
    - Use:
        - `Option<T>` when checking optional co-applicant.
        - `Result<T, E>` for loan eligibility errors (`AgeError`, `IncomeError`).
    - Implement `while let` and `match` for control flow.

- **Challenge**: Code must handle nested error types cleanly with `?` operator and propagate errors elegantly.

💾 Assignment 3: File Logger with Panic Handling (Real-World Safe Rust)

- **Topics covered**: File handling, Custom Traits, Panic handling (`abort` vs `unwind`), Memory Safety.
- **Task**:
    - Build a logger that writes to file.
    - On critical failure (disk full, permissions issue), `panic!`.
    - Show difference between `abort` and `unwind` behaviors.
    - Use trait for generalized logging (console/file).
- **Challenge**: Draw borrow checker flow chart for logger resource access.

5 Real-World Style Rust Assignments (Better & Deeper)

1️⃣ Secure Digital Wallet CLI (Ownership + Borrowing + Traits + Errors)

Goal: Build a command-line **digital wallet** that supports:

- User accounts (struct).
- Balance check & fund transfers.
- Password-based authentication using borrowing (`&str`).

- Central logging using trait objects (`dyn Logger`).
- Use `Result<T, E>` for errors like `IncorrectPassword`, `InsufficientFunds`.
- Handle transaction history using vector slices.

Why better:

- Mimics real crypto-wallet models.
- Enforces borrow checking in authentication.
- Combines traits, ownership, error handling.

2️⃣ File-Based Database (Vec, Slice, Box, RAII, Lifetimes)

Goal: Create a **lightweight key-value store** using:

- File-backed persistence (append-only).
- Records stored as heap-allocated `Box` structs.
- Access records using slices (`&[T]`).
- Use lifetime annotations to avoid data leaks.
- Auto-writeback on shutdown using RAII.

Advanced Twist: Visualize Box memory allocation with stack/heap diagrams after each operation.

Why better:

- Mimics real backend storage techniques.

- Combines heap allocations, slices, and lifetime control.
- Reinforces resource management without GC.

3️⃣ Real-Time Airline Booking Engine (Match, Pattern Guards, Option, Result)

Goal: Simulate an **airline seat reservation system**:

- Planes have a vector of seats (`Vec<Option<Seat>>`).
- Allocate/free seats using Option matching.
- Return custom errors using Result types.
- Use pattern guards (`if let Some(seat) = ...`).
- Handle overbooking via `panic!` and recover.

Advanced: Draw Option memory layout diagram vs C++ nullable pointer.

Why better:

- Models concurrency-safe resource allocation.
- Reinforces Rust's Option/Result power.

4️⃣ Banking Transaction Processor (Enum Dispatch + Trait Objects + Threads)

Goal: Multi-threaded transaction processor with:

- Transaction types as Enums (`Deposit`, `Withdraw`, `Transfer`).

- Trait object for processing (`dyn Processor`).
- Background thread that:
  - Reads transaction queue.
  - Executes and logs each using dynamic dispatch.
- Communicate using channels (`std::sync::mpsc`).

Why better:

- Practical for backend/microservices.
- Exposes ownership/borrowing in thread-safe contexts.
- Shows trait object usage vs static dispatch.

5️⃣ Memory-Safe In-Memory Cache with Expiry (RAII + HashMap + Drop Trait)

Goal: Build an **in-memory caching layer**:

- Key-value store using `HashMap`.
- Supports automatic expiry via timers.
- Implement custom cleanup logic using `Drop` trait (on cache shutdown).
- Demonstrate RAII in resource cleanup.
- Slice-based retrieval for efficient memory reads.