Creating a detailed schedule to build this language structure in 2-hour sessions each day can help you pace the development without feeling overwhelmed. Here's an ideal schedule for building this over the course of **4 weeks** (28 days). You can adjust the pace if you feel more comfortable or need to slow down.

Week 1: Initial Setup and Core Concepts

- **Day 1-2:** Setup the project environment, file structure, and basic definitions.
    - Create base files and folders for each section.
    - Define main categories like **CF**, **Concurrency**, **EH**, **Functions**, and **Types**.
- **Day 3-4: CF (Control Flow)** - Basic Structure and Branching
    - Implement the **Branching** section: break, continue, goto, return.
- **Day 5-6: CF (Control Flow)** - Conditional Statements
    - Work on **if**, **else**, **switch**, **ternary**.
- **Day 7: CF (Control Flow)** - Loops
    - Work on **for**, **foreach**, **while**.

Week 2: Expanding Control Flow and Concurrency

- **Day 8-9: Concurrency** - Asynchronous Concepts (Futures, Promises)
    - Implement **Futures** and **Promise** files under the **AAA** folder.
- **Day 10-11: Concurrency** - Threads (Creation, Pooling, Synchronization)
    - Work on basic threading features like creating threads, managing pooling, and synchronization.
- **Day 12-13: EH (Exceptions Handling)**
    - Implement **try**, **catch**, **throw** for error handling.
- **Day 14: Functions** - Declaration and Basic Structure
    - Work on function declaration, return types, and basic function body implementation.

Week 3: Functions, Memory Management, and Modules

- **Day 15-16: Functions** - Parameters and Invocation
    - Implement function parameter types (named, optional, positional) and **call/recursion** features.
- **Day 17-18: MAP (Modules, Abstractions, and Packages)**
    - Work on the **Imports** and **Exports** structures, handling absolute and relative imports.
- **Day 19-20: MM (Memory Management)** - Allocation & References
    - Implement **Heap** and **Stack** memory allocation files.
    - Work on **pointer** handling under the **References** section.
- **Day 21: MM (Memory Management)** - Deallocation
    - Implement **Garbage Collection** and **Manual** memory deallocation files.

Week 4: Objects, Libraries, Types, and Polishing

- **Day 22-23: Objects** - Encapsulation and Inheritance
  - Work on **Private/Public** encapsulation files.
  - Implement **Single**, **Multiple**, and **Interface** inheritance structures.
- **Day 24-25: Objects** - Polymorphism and Properties
  - Work on **overloading** and **overriding** methods.
  - Implement **Constructors**, **Fields**, and **Methods**.
- **Day 26: SL (Standard Libraries)** - Collections & IO
  - Implement **List**, **Queue**, **Set**, **Map**, and basic IO files (console, file, directory).
- **Day 27: SL (Standard Libraries)** - Serialization and Utilities
  - Implement serialization files (JSON, XML, TOML, etc.).
  - Add utility functions for **DateTime**, **Math**, **Random**, and **String**.
- **Day 28: Types** - Advanced, Composite, Primitives, and UD (User-Defined Types)
  - Work on **Advanced Types** (Function, Pointer, Nullable, Generic).
  - Implement **User-Defined Types** for **Class** and **Interface**.
  - Polish any final sections.

Time Breakdown Per Session

Each 2-hour session can be divided into:

- **30 minutes:** Overview of the topic and planning the next steps.
- **1 hour:** Working on code, creating structure, and implementing files.
- **30 minutes:** Reviewing the previous day's work, refining and testing the implementation.

Tips for Efficiency:

- Stick to the schedule as much as possible but be flexible if a section needs more or less time.
- Keep track of what you’ve accomplished daily (even small wins).
- Use the last few days to test the interactions between different sections, ensuring they work together seamlessly.
- After the 4 weeks, if needed, you can revisit and refine parts of the structure or add more features.

This 4-week schedule should make the task manageable, with a focus on maintaining steady progress while allowing time for revisions and adjustments. Does this sound feasible to you?