Assignment (Generic Currency Range) :

- **Problem Statement**
  - We want a system that manages **financial transactions** in different **currencies** (e.g., USD, EUR). Each currency has a **minimum** and **maximum** possible amount (for example, no negative amounts if that suits your business rules, or allow negatives if overdrafts are permissible). We need to:

    1. Represent a **generic** monetary amount (`Payment<TCurrency>`).
    2. Define **ranges** of permissible amounts (`PaymentRange<TSelf>`) with start and end values.
    3. Provide **containment**, **clamping**, and **intersection** operations on these ranges.
    4. Demonstrate usage via **manual iteration** over arrays/lists of data.

    No LINQ is allowed must use **loops** to process data.

1. Business Use Case

In a banking or e-commerce system, we might need to handle **payments** in multiple currencies. We:

- Ensure amounts don’t exceed certain **upper** or **lower** limits (like a maximum transaction limit).
- Check if a given payment is **valid** within a specific range (e.g., an order minimum or maximum).
- **Clamp** a payment that’s too high (or too low) into a safe range.

- Compute the **intersection** of two permissible ranges (e.g., one from regulatory limits, one from business policy).

2. Requirements & Constraints

1. `ICurrency` **Interface**
   - Has **static abstract** members: `double MinValue` and `double MaxValue`.
   - Each currency type implements this with domain-appropriate values (e.g., `USD`, `EUR`).
2. `Payment<TCurrency>` **Struct**
   - Constrained by `where TCurrency : ICurrency`.
   - Implements `IMinMaxValue<Payment<TCurrency>>` and `IComparable<Payment<TCurrency>>`.
   - Stores a `double Amount`.
   - Defines `MinValue`/`MaxValue` using the currency's `MinValue`/`MaxValue`.
   - Overrides `Equals`, `GetHashCode`, and implements `<`, `>`, etc.
   - Uses a **tolerance-based** comparison (e.g., `Math.Abs(this.Amount - other.Amount) < 0.0001`).
   - `ToString()` indicates the numeric amount plus the currency symbol/name.
3. `PaymentRange<TSelf>` **Class**
   - `where TSelf : struct, IComparable<TSelf>, IMinMaxValue<TSelf>`.
   - Read-only properties `Start` and `End`.
   - Constructor throws if `Start > End`.
   - Methods:
     - `bool Contains(TSelf value)`
     - `TSelf Clamp(TSelf value)`
     - `PaymentRange<TSelf> Intersect(PaymentRange<TSelf> other)`

- `ToString()` prints something like `"Range: {Start} to {End}"`.

4. **At Least Two Currencies**
   - For example, `USD` and `EUR`.
   - `MinValue` could be `0` if you disallow negatives, or `10000` if overdrafts are allowed.
   - `MaxValue` might be something large like `1e9`.
5. **No LINQ**
   - All array/list manipulations must be done with manual loops/conditionals.
6. **Console App Demonstration**
   - Show a series of examples (arrays/lists of payments) being iterated over.
   - Check containment in a range, clamp out-of-range values, compute intersections, etc.