In C#, `struct` and `record` are both ways to define types, but they differ in **value vs reference semantics**, **mutability**, and **intended use cases**.

Here's a **clear comparison**:

1. Memory & Type Semantics

Feature	`struct`	`record`
Type	Value type (stored on the stack or inline in containing type)	Reference type (stored on the heap) by default; `record struct` exists for value semantics
Copy Behavior	Copies the entire value when assigned or passed	Copies the reference (two variables point to the same object)
Default Mutability	Mutable (fields/properties can be changed)	Immutable by default (with `init`-only setters)
Nullability	Cannot be `null` unless declared as `Nullable<T>`	Can be `null` (unless `record struct`)

2. Equality

Feature	`struct`	`record`
Default Equality	Compares field-by-field values	Compares field-by-field values (value-based equality)
Customization	Must override `Equals` and `GetHashCode` manually for custom behavior	Automatically generated for you

3. Syntax Examples

Struct Example

```
public struct Point
{
    public int X { get; set; }
    public int Y { get; set; }
}

// Usage
var p1 = new Point { X = 1, Y = 2 };
var p2 = p1; // Creates a copy
p2.X = 5;    // Does not affect p1
```

Record Example

```
public record Person(string Name, int Age);

// Usage
var person1 = new Person("Alice", 30);
var person2 = person1 with { Age = 31 }; // Creates a new object
Console.WriteLine(person1 == person2);   // False (different values)
```

4. When to Use

- **Use** `struct` when:
  - You need a **lightweight value type** (e.g., coordinates, colors, small immutable data).
  - Performance matters and you want to avoid heap allocations.
  - The type is small (Microsoft recommends ≤ 16 bytes for efficiency).
- **Use** `record` when:
  - You need **immutable reference types** with **value-based equality**.
  - You want concise syntax for data models (especially in functional or DTO scenarios).
  - You want to use `with` expressions for non-destructive mutation.

✅ **Tip:**
C# also supports `record struct` (value type with record features) and `readonly record struct` (immutable value type with record features), giving you the best of both worlds.

If you want, I can prepare a **side-by-side runnable C# demo** showing how `struct`, `record`, and `record struct` behave differently in memory and equality.
Would you like me to do that?