Use a struct when:

- You need a value type: Structs are stored directly where they are declared (on the stack for local variables, or inline within other objects), which can improve performance by reducing garbage collector pressure.
- The type logically represents a single value: Examples include coordinates, colors, or numeric ranges.
- The size is small: Microsoft recommends a size of 16 bytes or less for optimal performance . Large structs can be inefficient to pass by value.
- The type is immutable (preferred): Although structs can be mutable, immutable structs are generally safer and easier to work with, avoiding unexpected side effects when passed as arguments .
- You don't need inheritance: Structs cannot inherit from other structs or classes, and they are sealed (cannot be inherited from) .
- You don't need nullability (initially): Value types cannot be null by default, but you can use Nullable types.

Use record when

Records, introduced in C# 9, are designed to simplify the creation of immutable data models with less boilerplate code.

- Immutable Data Models: The primary use case for records is data that should not change after creation. This ensures thread safety in concurrent environments.
- Value-Based Equality: Two record instances are considered equal if all of their public property values match, regardless of their memory location (unlike classes, which use reference equality by default). This is useful for comparison and storage in hash-based collections where you need the hash code to remain constant.
- Data Transfer Objects (DTOs): Records are ideal for passing simple data structures between different application layers or services, such as in REST APIs or microservices.
- Value Objects in DDD: In Domain-Driven Design, value objects are immutable and compared by their values, making records a natural fit.
- Concise Syntax: Records provide a concise syntax for defining properties and automatically generating useful methods like Equals(), GetHashCode(), and a formatted ToString().

- Nondestructive Mutation: The with expression allows you to create a new record instance that is a copy of an existing one, but with specified properties modified, leaving the original object untouched.