C# structure:

Introduction:

Structure is a value type and a collection of variables of different data types under a single unit. It is almost similar to a class because both are user defined data types and both hold a bunch of different data types. C# provide the availability to use predefined data types. However, sometimes the user might be in need to define its own data types which are also known as user defined data types. Although it comes under the value type the user can modify it according to requirements and that’s why it is also termed as user-defined data type.

- **Defining structure:**

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member for your program.

Using struct keyword one can defined the structure consisting different datatypes in it. A structure can also contain constructor, constant, field methods, properties, indexer and event etc.

Syntax:

```
Access-modifier struct structure-name{
// fields
//parameterized constructor
//constant
//properties
//indexers
//events
//methode etc.
}
```

Ex1:

```
Public struct User{
        public string name;
        Public string location;
        Public int age;
}
```

If you observe above example, we defined a structure called "User" with required fields and we can add required methods and properties based on our requirements.

- **C# structure initialization:**

In c#, structure can be instantiated with or without **new** keyword.

i.e

```
User U1= new User();
User U2;

U1.name="hari";
U1.location="PKR";
U1.age=32;
U2=U1;
```

- **C# structure with constructor:**

In c#, the structures won't allow us to declare a default constructor or a constructor without parameters. It won't allows us to initialize fields with values unless they are declared as **const** or **static.** Following is the example of defining a structure with parametrized constructor and initializing the constructor fields in c# programming language.

- **Ex of c# structor with constructor**

```
public struct User{
    public string name,location;
    public User(string a, string b) // parameterized constructor
    {
        name=a;
        location=b;
        }
}
```

Ex2:

```
using System;

struct Books {
   public string title;
   public string author;
   public string subject;
   public int book_id;
};

public class testStructure {
   public static void Main(string[] args) {
      Books Book1;   /* Declare Book1 of type Book */
      Books Book2 = new Books();   // Declare Book2 of type Book

      /* book 1 specification */
      Book1.title = "C Programming";
      Book1.author = "Nuha Ali";
      Book1.subject = "C Programming Tutorial";
      Book1.book_id = 6495407;

      /* book 2 specification */
      Book2.title = "Telecom Billing";
      Book2.author = "Zara Ali";
      Book2.subject =  "Telecom Billing Tutorial";
      Book2.book_id = 6495700;

      /* print Book1 info */
      Console.WriteLine( "Book 1 title : {0}", Book1.title);
      Console.WriteLine("Book 1 author : {0}", Book1.author);
      Console.WriteLine("Book 1 subject : {0}", Book1.subject);
      Console.WriteLine("Book 1 book_id :{0}", Book1.book_id);

      /* print Book2 info */
      Console.WriteLine("Book 2 title : {0}", Book2.title);
      Console.WriteLine("Book 2 author : {0}", Book2.author);
      Console.WriteLine("Book 2 subject : {0}", Book2.subject);
      Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);

      Console.ReadKey();
    }
  }
```

- **When the above code is compiled and executed, it produces the following result –**

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

- **Advantages of Using Structs**

1. Structs are value types, which means they are stored on the stack and are more memory-efficient than reference types like classes.
2. They offer better performance in certain scenarios due to their value type nature and reduced garbage collection overhead.
3. Structs can be used as a safer alternative to primitive types by encapsulating related data and providing relevant methods.

- **Features of C# Structures**
  - ❖ Structures can have methods, fields, indexers, properties, operator methods, and events.

  - ❖ Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.

  - ❖ Unlike classes, structures cannot inherit other structures or classes.

  - ❖ Structures cannot be used as a base for other structures or classes.

  - ❖ A structure can implement one or more interfaces.

  - ❖ Structure members cannot be specified as abstract, virtual, or protected.

  - ❖ When you create a struct object using the New operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the New operator.

  - ❖ If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.
  - ❖ In c#, the structures won’t allow us to declare a default constructor or a constructor without parameters. It won’t allows us to initialize fields with values unless they are declared as const or static.

- **Difference between class and structure**

Class	**Structure**
◆ It is reference type	◆ It is value type.
◆ It’s object is created on heap memory.	◆ Its object(structure’s variable) is created on a stack memory.
◆ Its supports inheritance.	◆ It doesn't support inheritance.
◆ It can have all type of constructor and destructor.	◆ It can have only parametrized constructor and don’t have destructor.
◆ To define class we use ‘class’ keyword.	◆ To define a structure we use ‘struct’ keyword.
◆ To create an object for class we have to use new operator.	◆ To create an object for structure we don’t have to use new operator.