

C# Structures (Structs)

In **c#**, **structures** are same as classes (/tutorial/csharp/csharp-classes-and-objects-with-examples), but the only difference is classes (/tutorial/csharp/csharp-classes-and-objects-with-examples) are the reference types (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples), and structures are the value types (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples). As a value type (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples), the structures directly contain their value, so their object or instance is stored on the stack, and structures are faster than classes (/tutorial/csharp/csharp-classes-and-objects-with-examples).

In **c#**, the structures can contain fields, properties, member functions, operators (/tutorial/csharp/csharp-operators-arithmetic-relational-logical-assignment-precedence), constructors (/tutorial/csharp/csharp-constructors-with-examples), events, indexers, constants (/tutorial/csharp/csharp-const-constant-keyword), and even other structure types.

Create Structures in C#

In **c#**, structures can be created by using `struct` keyword. Following is the declaration of structure in the **c#** programming language.

```
public struct users
{
    // Properties, Methods, Events, etc.
}
```

If you observe the above syntax, we defined a structure "**users**" using `struct` keyword with **public** access modifier. Here, the **public** access specifier will allow the users to create objects for this structure, and inside of the body structure, we can create required fields, properties, methods, and events to use in our applications.

Following is the example of defining a structure in the **c#** programming language.

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

If you observe the 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#**, structures can be instantiated with or without `new` keyword. Following is the example of assigning values to the variables of structure.

```
User u = new User();
u.name = "Suresh Dasari";
```

```
u.location = "Hyderabad";  
u.age = 32;
```

To make use of fields, methods, and events of structure, then it's mandatory to instantiate a structure with `new` keyword in c# programming language.

C# Structure with Constructor

In c#, the structures won't allow us to declare a default constructor (/tutorial/csharp/csharp-constructors-with-examples) or a constructor (/tutorial/csharp/csharp-constructors-with-examples) without parameters. It won't allow us to initialize fields with values unless they are declared as `const` (/tutorial/csharp/csharp-const-constant-keyword) or `static` (/tutorial/csharp/csharp-static-keyword).

Following is the example of defining a structure with parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzpst) and initializing the constructor (/tutorial/csharp/csharp-constructors-with-examples) fields in the c# programming language.

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

If you observe the above example, we defined a structure called "**User**" with required fields and parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzpst).

C# Structure with Default Constructor

As discussed, the structures will allow only parameterized constructors (/tutorial/csharp/csharp-constructors-with-examples#divcspzpst), and fields cannot be initialized unless they are declared as `const` (/tutorial/csharp/csharp-const-constant-keyword) or `static` (/tutorial/csharp/csharp-static-keyword).

Following is the example of defining a structure with a default constructor (/tutorial/csharp/csharp-constructors-with-examples) and initializing fields with values.

```
struct User  
{  
    // Compile error  
    public string name = "Suresh Dasari";  
    public string location;  
    public int age;  
    // Compile error  
    public User()  
    {  
        location = "Hyderabad";  
        age = 32;  
    }  
}
```

```
}  
}
```

When we execute the above code, we will get compiler errors because we declared a structure with the default constructor (/tutorial/csharp/csharp-constructors-with-examples) (parameterless) and initialized fields without defining it as `const` (/tutorial/csharp/csharp-const-constant-keyword) or `static` (/tutorial/csharp/csharp-static-keyword).

In `c#`, if we create a structure with the parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzcst), then we must need to explicitly initialize all the fields within the constructor (/tutorial/csharp/csharp-constructors-with-examples) before the control is returned to the caller; otherwise, we will get a compile-time error.

Following is the example of defining a structure with the parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzcst) and required fields in the `c#` programming language.

```
public struct User  
{  
    public string name, location;  
    // Compile Error  
    public User(string a)  
    {  
        name = a;  
    }  
}
```

When you execute the above code, we will get a compile-time error because we defined **name** and **location** variables, but we assign a value to the only **name** variable.

As discussed, if we create a structure with the parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzcst), then we must need to explicitly initialize all the fields before leaving the constructor (/tutorial/csharp/csharp-constructors-with-examples).

Now we will see how to create a structure with fields and parameterized constructors (/tutorial/csharp/csharp-constructors-with-examples#divcspzcst) in `c#` programming language with example.

C# Structure Example

Following is the example of creating a structure with different fields and parameterized constructors (/tutorial/csharp/csharp-constructors-with-examples#divcspzcst) in `c#` programming language with various data members and member functions.

```
using System;  
  
namespace Tutlane  
{  
    struct User  
    {  
        public const string name = "Suresh Dasari";  
        public string location;  
        public int age;  
        public User(string a, int b)  
        {  

```

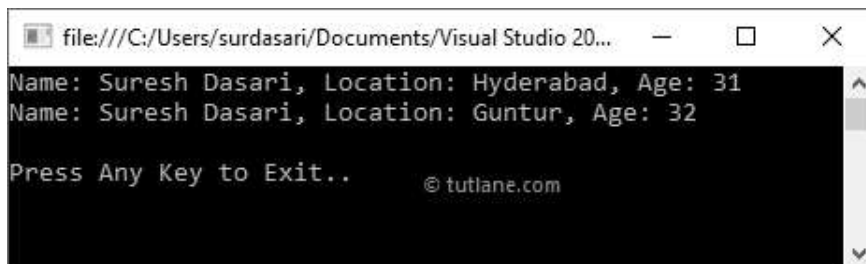
```

        location = a;
        age = b;
    }
}
class Program
{
    static void Main(string[] args)
    {
        // Declare object with new keyword
        User u = new User("Hyderabad", 31);
        // Declare object without new keyword
        User u1;
        Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, u.location, u.age);
        // Initialize Fields
        u1.location = "Guntur";
        u1.age = 32;
        Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, u1.location, u1.age);
        Console.WriteLine("\nPress Enter Key to Exit..");
        Console.ReadLine();
    }
}

```

If you observe the above example, we defined a structure (**User**) by including required fields, parameterized constructor (/tutorial/csharp/csharp-constructors-with-examples#divcspzpst), and created an instance of structure (**User**) with and without `new` keyword to initialize or get field values based on our requirements.

When you execute the above c# program, you will get the result as shown below.



```

file:///C:/Users/surdasari/Documents/Visual Studio 20...
Name: Suresh Dasari, Location: Hyderabad, Age: 31
Name: Suresh Dasari, Location: Guntur, Age: 32
Press Any Key to Exit..
© tutlane.com

```

In c#, by using structures we can implement an interface (/tutorial/csharp/csharp-interface), but structures cannot inherit from another structure or class.

C# Structure Characteristics

The following are the important characteristics of structures in the c# programming language.

- In c#, structures are value types (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples), and those are defined by using `struct` keyword.
- During the structure declaration, the fields cannot be initialized unless defined as `const` (/tutorial/csharp/csharp-const-constant-keyword) or `static` (/tutorial/csharp/csharp-static-keyword).
- Structures in c# can include fields, properties, member functions, operators, constructors, events, indexers, constants, and even other structure types.

- Structures cannot include the default constructor (/tutorial/csharp/csharp-constructors-with-examples) (constructor without parameters) or destructor (/tutorial/csharp/csharp-destructor-with-examples), but it will allow us to declare constructors with parameters (/tutorial/csharp/csharp-destructor-with-examples).
- A structure cannot inherit from another structure or class (/tutorial/csharp/csharp-classes-and-objects-with-examples).
- In c#, the structure can implement interfaces (/tutorial/csharp/csharp-interface).
- A structure can be instantiated with or without using a `new` keyword.

C# Difference between Structure and Class

The following are the difference between structures and classes in the c# programming language.

- In c#, classes (/tutorial/csharp/csharp-classes-and-objects-with-examples) are the reference types (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples), and structures are the value types (/tutorial/csharp/csharp-value-type-and-reference-type-with-examples).
- Classes (/tutorial/csharp/csharp-classes-and-objects-with-examples) can contain default constructors (/tutorial/csharp/csharp-constructors-with-examples) or destructors (/tutorial/csharp/csharp-destructor-with-examples), but structures will contain only constructors that have parameters.
- We can implement inheritance (/tutorial/csharp/csharp-inheritance) using Classes (/tutorial/csharp/csharp-classes-and-objects-with-examples), but structures won't support inheritance (/tutorial/csharp/csharp-inheritance).
- Unlike classes (/tutorial/csharp/csharp-classes-and-objects-with-examples), structs can be instantiated with or without using a `new` operator.

CONTACT US

📍 **Address:** No.1-93, Pochamma Colony, Manikonda, Hyderabad, Telangana - 500089

✉ **Email:** support@tutlane.com (mailto:support@tutlane.com)