C# - Structures

[Previous Page](https://www.tutorialspoint.com/csharp/csharp_strings.htm)

[Next Page](https://www.tutorialspoint.com/csharp/csharp_enums.htm)

In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The **struct** keyword is used for creating a structure.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

* Title
* Author
* Subject
* Book ID

Defining a 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.

For example, here is the way you can declare the Book structure −

struct Books {

public string title;

public string author;

public string subject;

public int book\_id;

};

The following program shows the use of the structure −

[Live Demo](http://tpcg.io/PCFGlU)

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; /\* 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

Features of C# Structures

You have already used a simple structure named Books. Structures in C# are quite different from that in traditional C or C++. The C# structures have the following features −

* 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.

Class versus Structure

Classes and Structures have the following basic differences −

* classes are reference types and structs are value types
* structures do not support inheritance
* structures cannot have default constructor

In the light of the above discussions, let us rewrite the previous example −

[Live Demo](http://tpcg.io/VJUEs4)

using System;

struct Books {

private string title;

private string author;

private string subject;

private int book\_id;

public void getValues(string t, string a, string s, int id) {

title = t;

author = a;

subject = s;

book\_id = id;

}

public void display() {

Console.WriteLine("Title : {0}", title);

Console.WriteLine("Author : {0}", author);

Console.WriteLine("Subject : {0}", subject);

Console.WriteLine("Book\_id :{0}", book\_id);

}

};

public class testStructure {

public static void Main(string[] args) {

Books Book1 = new Books(); /\* Declare Book1 of type Book \*/

Books Book2 = new Books(); /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

Book1.getValues("C Programming",

"Nuha Ali", "C Programming Tutorial",6495407);

/\* book 2 specification \*/

Book2.getValues("Telecom Billing",

"Zara Ali", "Telecom Billing Tutorial", 6495700);

/\* print Book1 info \*/

Book1.display();

/\* print Book2 info \*/

Book2.display();

Console.ReadKey();

}

}

When the above code is compiled and executed, it produces the following result −

Title : C Programming

Author : Nuha Ali

Subject : C Programming Tutorial

Book\_id : 6495407

Title : Telecom Billing

Author : Zara Ali

Subject : Telecom Billing Tutorial

Book\_id : 6495700

C# - Enums

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.

C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring *enum* Variable

The general syntax for declaring an enumeration is −

enum <enum\_name> {

enumeration list

};

Where,

* The *enum\_name* specifies the enumeration type name.
* The *enumeration list* is a comma-separated list of identifiers.

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example −

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

Example

The following example demonstrates use of enum variable −

[Live Demo](http://tpcg.io/JkqN9p)

using System;

namespace EnumApplication {

class EnumProgram {

enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

static void Main(string[] args) {

int WeekdayStart = (int)Days.Mon;

int WeekdayEnd = (int)Days.Fri;

Console.WriteLine("Monday: {0}", WeekdayStart);

Console.WriteLine("Friday: {0}", WeekdayEnd);

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

Monday: 1

Friday: 5