C# is a simple & powerful object-oriented programming language developed by Microsoft. C# can be used to create various types of applications, such as web, windows, console applications, or other types of applications using Visual studio.

C# Version History

C# was first introduced with .NET Framework 1.0 in the year 2002 and evolved much since then. The following table lists important features introduced in each version of C#:

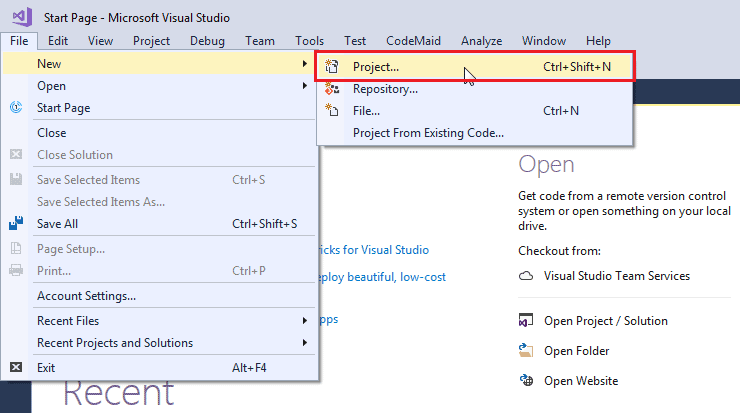
| Version | .NET Framework | Visual Studio | Important Features |
| --- | --- | --- | --- |
| C# 1.0 | .NET Framework 1.0/1.1 | Visual Studio .NET 2002 | * Basic features |
| C# 2.0 | .NET Framework 2.0 | Visual Studio 2005 | * Generics * Partial types * Anonymous methods * Iterators * Nullable types * Private setters (properties) * Method group conversions (delegates) * Covariance and Contra-variance * Static classes |
| C# 3.0 | .NET Framework 3.0\3.5 | Visual Studio 2008 | * Implicitly typed local variables * Object and collection initializers * Auto-Implemented properties * Anonymous types * Extension methods * Query expressions * Lambda expressions * Expression trees * Partial Methods |
| C# 4.0 | .NET Framework 4.0 | Visual Studio 2010 | * Dynamic binding (late binding) * Named and optional arguments * Generic co- and contravariance * Embedded interop types |
| C# 5.0 | .NET Framework 4.5 | Visual Studio 2012/2013 | * Async features * Caller information |
| C# 6.0 | .NET Framework 4.6 | Visual Studio 2013/2015 | * Expression Bodied Methods * Auto-property initializer * nameof Expression * Primary constructor * Await in catch block * Exception Filter * String Interpolation |
| C# 7.0 | .NET Core 2.0 | Visual Studio 2017 | * out variables * Tuples * Discards * Pattern Matching * Local functions * Generalized async return types |
| C# 8.0 | .NET Core 3.0 | Visual Studio 2019 | * Readonly members * Default interface methods * Using declarations * Static local functions * Disposable ref structs * Nullable reference types |
| C# 9.0 | .NET 5.0 | Visual Studio 2019 | * Records * Init-only properties * Top-level statements * Init accessors and readonly fields * With-expressions * Value-based equality |
| C# 10.0 | .NET 6.0 | Visual Studio 2022 | * Record structs * Global using directives * File-scoped namespace declaration * Extended Proptery Patterns * Null Parameter Checking * Constant interpolated strings |

First C# Program

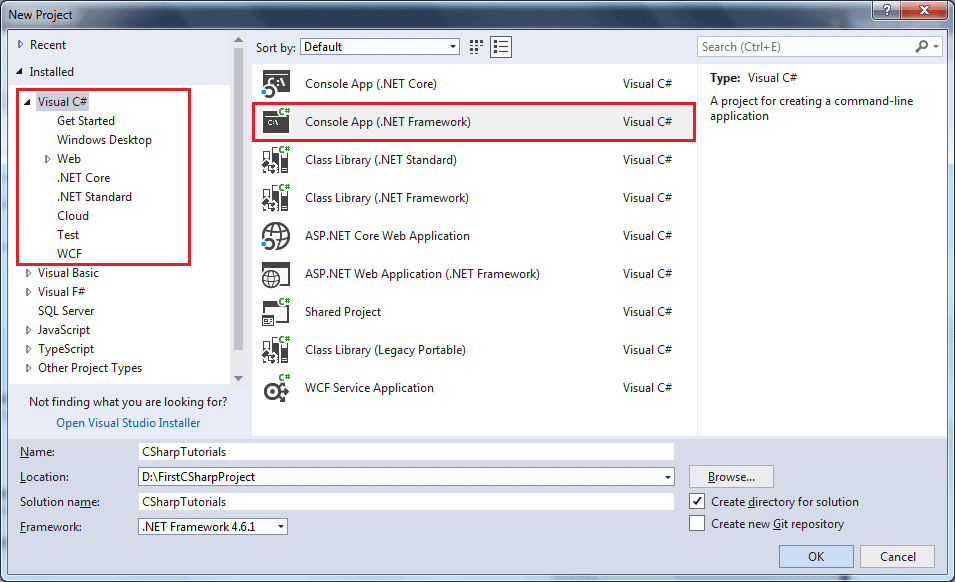
Here, you will learn to create a simple console application in C# and understand the basic building blocks of a console application.

C# can be used in a window-based, web-based, or console application. To start with, we will create a console application to work with C#.

Open Visual Studio (2017 or later) installed on your local machine. Click on File -> New Project... from the top menu, as shown below.

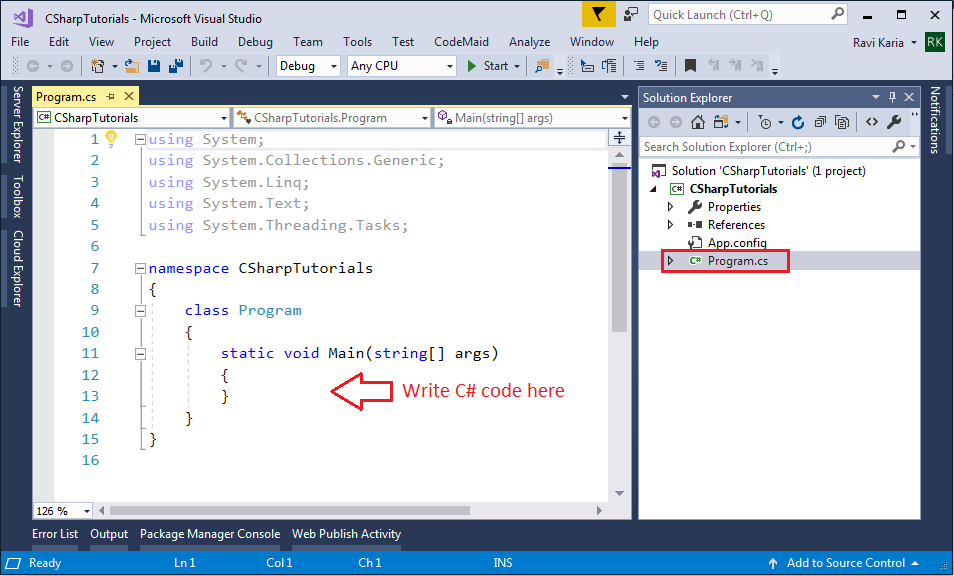
[](https://www.tutorialsteacher.com/Content/images/csharp/create-project-in-visualstudio.png)Create a New Project in Visual Studio 2017

From the **New Project** popup, shown below, select Visual C# in the left side panel and select the Console App in the right-side panel.

[](https://www.tutorialsteacher.com/Content/images/csharp/create-csharp-console-project.png)Select Visual C# Console App Template

In the name section, give any appropriate project name, a location where you want to create all the project files, and the name of the project solution.

Click OK to create the console project. **Program.cs** will be created as default a C# file in Visual Studio where you can write your C# code in Program class, as shown below. (The .cs is a file extension for C# file.)

[](https://www.tutorialsteacher.com/Content/images/csharp/csharp-program.png)C# Console Program

Every console application starts from the Main() method of the Program class. The following example displays "Hello World!!" on the console.

Example: C# Console Application

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSharpTutorials

{

class Program

{

static void Main(string[] args)

{

string message = "Hello World!!";

Console.WriteLine(message);

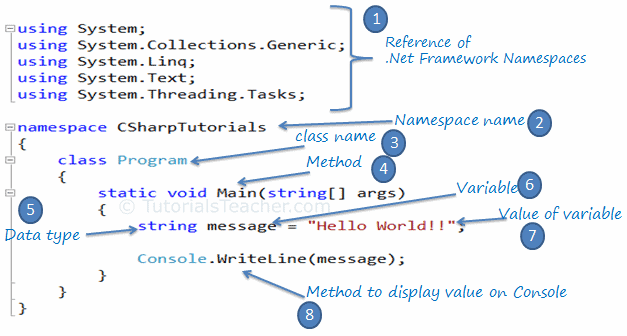
}

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ulc73T)

The following image illustrates the important parts of the above example.

[](https://www.tutorialsteacher.com/Content/images/csharp/csharp-code-structure.png)C# Code Structure

Let's understand the above C# structure.

1. Every .NET application takes the reference of the necessary .NET framework namespaces that it is planning to use with the using keyword, e.g., using System.Text.
2. Declare the namespace for the current class using the namespace keyword, e.g., namespace CSharpTutorials.FirstProgram
3. We then declared a class using the class keyword: class Program
4. The Main() is a method of Program class is the entry point of the console application.
5. String is a data type.
6. A message is a [variable](https://www.tutorialsteacher.com/csharp/csharp-variable) that holds the value of a specified [data type](https://www.tutorialsteacher.com/csharp/csharp-data-types).
7. "Hello World!!" is the value of the message variable.
8. The Console.WriteLine() is a static method, which is used to display a text on the console.

 Note:

Every line or statement in C# must end with a semicolon (;).

Compile and Run C# Program

To see the output of the above C# program, we have to compile it and run it by pressing Ctrl + F5 or clicking the Run button or by clicking the "Debug" menu and clicking "Start Without Debugging". You will see the following output in the console:

Output:

Hello World!!

So this is the basic code items that you will probably use in every C# code.

C# Keywords

C# contains reserved words that have special meaning for the compiler. These reserved words are called "keywords". Keywords cannot be used as an identifier (name of a variable, class, interface, etc.).

Keywords in C# are distributed under the following categories:

Modifier Keywords

Modifier keywords are specific keywords that indicate who can modify types and type members. Modifiers allow or prevent certain parts of programs from being modified by other parts.

| Modifier keywords |
| --- |
| abstract |
| async |
| const |
| [event](https://www.tutorialsteacher.com/csharp/csharp-event) |
| extern |
| new |
| override |
| [partial](https://www.tutorialsteacher.com/csharp/csharp-partial-class) |
| readonly |
| sealed |
| [static](https://www.tutorialsteacher.com/csharp/csharp-static) |
| unsafe |
| virtual |
| volatile |

Access Modifier Keywords:

Access modifiers are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members.

| Access Modifiers | Usage |
| --- | --- |
| public | The Public modifier allows any part of the program in the same assembly or another assembly to access the type and its members. |
| private | The Private modifier restricts other parts of the program from accessing the type and its members. Only code in the same class or struct can access it. |
| internal | The Internal modifier allows other program code in the same assembly to access the type or its members. This is default access modifiers if no modifier is specified. |
| protected | The Protected modifier allows codes in the same class or a class that derives from that class to access the type or its members. |

Statement Keywords

Statement keywords are related to program flow.

| Statement Keywords |
| --- |
| [if](https://www.tutorialsteacher.com/csharp/csharp-if-else) |
| [else](https://www.tutorialsteacher.com/csharp/csharp-if-else) |
| [switch](https://www.tutorialsteacher.com/csharp/csharp-switch) |
| case |
| [do](https://www.tutorialsteacher.com/csharp/csharp-do-while-loop) |
| [for](https://www.tutorialsteacher.com/csharp/csharp-for-loop) |
| foreach |
| in |
| [while](https://www.tutorialsteacher.com/csharp/csharp-while-loop) |
| break |
| continue |
| default |
| goto |
| return |
| yield |
| [throw](https://www.tutorialsteacher.com/csharp/throw-csharp) |
| try |
| catch |
| finally |
| checked |
| unchecked |
| fixed |
| lock |

Method Parameter Keywords

These keywords are applied to the parameters of a method.

| Method Parameter Keywords |
| --- |
| params |
| ref |
| [out](https://www.tutorialsteacher.com/articles/out-keyword-in-csharp) |

Namespace Keywords

These keywords are applied with namespace and related operators.

| Namespace Keywords |
| --- |
| using |
| . operator |
| :: operator |
| extern alias |



Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Operator Keywords

Operator keywords perform miscellaneous actions.

| Operator Keywords |
| --- |
| as |
| await |
| is |
| new |
| sizeof |
| typeof |
| stackalloc |
| checked |
| unchecked |

Access Keywords

Access keywords are used to access the containing class or the base class of an object or class.

| Access keywords |
| --- |
| base |
| this |

Literal Keywords

Literal keywords apply to the current instance or value of an object.

| Literal Keywords |
| --- |
| null |
| false |
| true |
| value |
| void |

Type Keywords

Type keywords are used for data types.

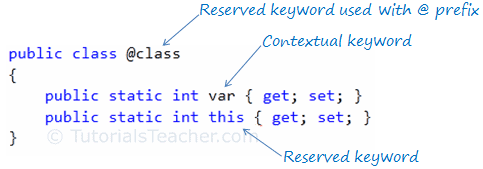
| Type keywords |
| --- |
| bool |
| byte |
| char |
| class |
| decimal |
| double |
| enum |
| float |
| int |
| long |
| sbyte |
| short |
| string |
| struct |
| uint |
| ulong |
| ushort |

Contextual Keywords

Contextual keywords are considered as keywords, only if used in specific contexts. They are not reserved and so can be used as names or identifiers.

| Contextual Keywords |
| --- |
| add |
| var |
| dynamic |
| global |
| set |
| value |

Contextual keywords are not converted into blue color (default color for keywords in visual studio) when used as an identifier in Visual Studio. For example, var in the below figure is not in blue, whereas the color of this is the blue color. So var is a contextual keyword.

[](https://www.tutorialsteacher.com/Content/images/csharp/keywords-in-vs.png)C# Keywords

Query Keywords

Query keywords are contextual keywords used in LINQ queries.

| Query Keywords |
| --- |
| from |
| where |
| select |
| group |
| into |
| orderby |
| join |
| let |
| in |
| on |
| equals |
| by |
| ascending |
| descending |

As mentioned above, a keyword cannot be used as an identifier (name of the variable, class, interface, etc.). However, they can be used with the prefix '@'. For example, the class is a reserved keyword, so it cannot be used as an identifier, but @class can be used as shown below.

Example: Use Keyword as Identifier

public class @class

{

public static int MyProperty { get; set; }

}

@class.MyProperty = 100;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GoQsN9)

C# Class

A class is like a blueprint of a specific object. In the real world, every object has some color, shape, and functionalities - for example, the luxury car Ferrari. Ferrari is an object of the luxury car type. The luxury car is a class that indicates some characteristics like speed, color, shape, interior, etc. So any company that makes a car that meets those requirements is an object of the luxury car type. For example, every single car of BMW, Lamborghini, Cadillac are an object of the class called 'Luxury Car'. Here, 'Luxury Car' is a class, and every single physical car is an object of the luxury car class.

Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.

A class enables you to create your custom types by grouping variables of other types, methods, and events.

In C#, a class can be defined by using the class keyword.

Example: C# Class

public class MyClass

{

public string myField = string.Empty;

public MyClass()

{

}

public void MyMethod(int parameter1, string parameter2)

{

Console.WriteLine("First Parameter {0}, second parameter {1}",

parameter1, parameter2);

}

public int MyAutoImplementedProperty { get; set; }

private int myPropertyVar;

public int MyProperty

{

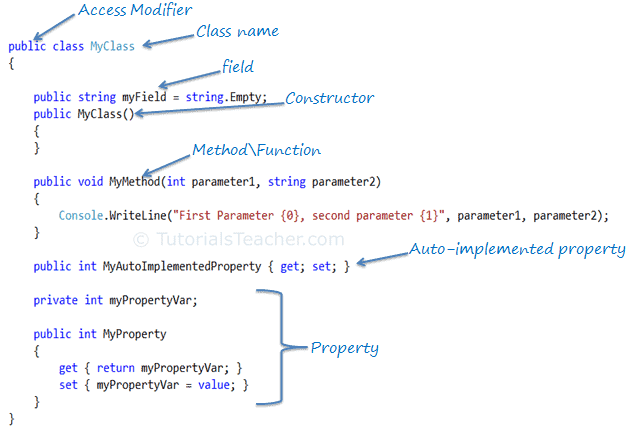
get { return myPropertyVar; }

set { myPropertyVar = value; }

}

}

The following image shows the important building blocks of C# class.

[](https://www.tutorialsteacher.com/Content/images/csharp/csharp-class.png)C# Class

C# Access Modifiers

Access modifiers are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members. Public, private, protected, and internal are access modifiers in C#. We will learn about it in the [keyword](https://www.tutorialsteacher.com/csharp/csharp-keywords) section.

C# Field

The field is a class-level variable that holds a value. Generally, field members should have a private access modifier and used with property.

C# Constructor

A class can have parameterized or parameterless constructors. The constructor will be called when you create an instance of a class. Constructors can be defined by using an access modifier and class name: <access modifiers> <class name>(){ }

Example: Constructor in C#

class MyClass

{

public MyClass()

{

}

}

C# Method

A method can be defined using the following template:

{access modifier} {return type} MethodName({parameterType parameterName})

Example: Method in C#

public void MyMethod(int parameter1, string parameter2)

{

// write your method code here..

}

Property

A property can be defined using getters and setters, as shown below:

Example: Property in C#

private int \_myPropertyVar;

public int MyProperty

{

get { return \_myPropertyVar; }

set { \_myPropertyVar = value; }

}

Property encapsulates a private field. It provides getters (get{}) to retrieve the value of the underlying field and setters (set{}) to set the value of the underlying field. In the above example, \_myPropertyVar is a private field that cannot be accessed directly. It will only be accessed via MyProperty. Thus, MyProperty encapsulates \_myPropertyVar.

You can also apply some additional logic in get and set, as in the below example.

Example: Property in C#

private int \_myPropertyVar;

public int MyProperty

{

get {

return \_myPropertyVar / 2;

}

set {

if (value > 100)

\_myPropertyVar = 100;

else

\_myPropertyVar = value; ;

}

}

Auto-implemented Property

From C# 3.0 onwards, property declaration has been made easy if you don't want to apply some logic in get or set.

The following is an example of an auto-implemented property:

Example: Auto implemented property in C#

public int MyAutoImplementedProperty { get; set; }

Notice that there is no private backing field in the above property example. The backing field will be created automatically by the compiler. You can work with an automated property as you would with a normal property of the class. Automated-implemented property is just for easy declaration of the property when no additional logic is required in the property accessors.

Namespace

The namespace is a container for a set of related classes and namespaces. The namespace is also used to give unique names to classes within the namespace name. Namespace and classes are represented using a dot (.).

In C#, namespace can be defined using the namespace keyword.

Example: Namespace

namespace CSharpTutorials

{

class MyClass

{

}

}

In the above example, the fully qualified class name of MyClass is CSharpTutorials.MyClass.

A namespace can contain other namespaces. Inner namespaces can be separated using (.).

Example: Namespace

namespace CSharpTutorials.Examples

{

class MyClassExample

{

}

}

In the above example, the fully qualified class name of MyClassExample is CSharpTutorials.Examples.MyClassExample.

C# Variables

In C#, a variable contains a data value of the specific [data type](https://www.tutorialsteacher.com/csharp/csharp-data-types).

Syntax

<data type> <variable name> = <value>;

The following declares and initializes a variable of an int type.

Example: C# Variable

int num = 100;

Above, int is a data type, num is a variable name (identifier). The = operator is used to assign a value to a variable. The right side of the = operator is a value that will be assigned to left side variable. Above, 100 is assigned to a variable num.

The following declares and initializes variables of different data types.

Example: C# Variables

int num = 100;

float rate = 10.2f;

decimal amount = 100.50M;

char code = 'C';

bool isValid = true;

string name = "Steve";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ulc73T)

The followings are naming conventions for declaring variables in C#:

* Variable names must be unique.
* Variable names can contain letters, digits, and the underscore \_ only.
* Variable names must start with a letter.
* Variable names are case-sensitive, num and Num are considered different names.
* Variable names cannot contain reserved keywords. Must prefix @ before keyword if want reserve keywords as identifiers.

C# is the strongly typed language. It means you can assign a value of the specified data type. You cannot assign an integer value to string type or vice-versa.

Example: Cannot assign string to int type variable

int num = "Steve";

Variables can be declared first and initialized later.

Example: Late Initialization

int num;

num = 100;

A variable must be assigned a value before using it, otherwise, C# will give a compile-time error.

Error: Invalid Assignment

int i;

int j = i; //compile-time error: Use of unassigned local variable 'i'

The value of a variable can be changed anytime after initializing it.

Example: C# Variable

int num = 100;

num = 200;

Console.WriteLine(num); //output: 200

Multiple variables of the same data type can be declared and initialized in a single line separated by commas.

Example: Multiple Variables in a Single Line

int i, j = 10, k = 100;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ivh807)

Multiple variables of the same type can also be declared in multiple lines separated by a comma. The compiler will consider it to be one statement until it encounters a semicolon ;.

Example: Multi-Line Declarations

int i = 0,

j = 10,

k = 100;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-mzidSt)

The value of a variable can be assigned to another variable of the same data type. However, a value must be assigned to a variable before using it.

Example: Variable Assignment

int i = 100;

int j = i; // value of j will be 100

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-bg29Zx)

In C#, variables are categorized based on how they store their value in memory. Variables can be [value type or reference type](https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type) or pointer type.

C# - var

In C#, variables must be declared with the data type. These are called explicitly typed variables.

Example: Explicitly Typed Variable

int i = 100;// explicitly typed variable

C# 3.0 introduced var keyword to declare method level variables without specifying a data type explicitly.

Example: Implicitly Typed Local Variable

var j = 100; // implicitly typed local variable

The compiler will infer the type of a variable from the expression on the right side of the = operator. Above, var will be compiled as int.

The following infers the type from an expression.

Example: var from expression

int i = 10;

var j = i + 1; // compiles as int

var can be used to declare any built-in data type or a user-defined type or an anonymous type variable. The following example shows C# compiler infers type based on the value:

Example: Implicitly-Typed Variable

static void Main(string[] args)

{

var i = 10;

Console.WriteLine("Type of i is {0}", i.GetType());

var str = "Hello World!!";

Console.WriteLine("Type of str is {0}", str.GetType());

var dbl = 100.50d;

Console.WriteLine("Type of dbl is {0}", dbl.GetType());

var isValid = true;

Console.WriteLine("Type of isValid is {0}", isValid.GetType());

var ano = new { name = "Steve" };

Console.WriteLine("Type of ano is {0}", ano.GetType());

var arr = new[] { 1, 10, 20, 30 };

Console.WriteLine("Type of arr is {0}", arr.GetType());

var file = new FileInfo("MyFile");

Console.WriteLine("Type of file is {0}", file.GetType());

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-1JziVB)

Implicitly-typed variables must be initialized at the time of declaration; otherwise C# compiler would give an error: Implicitly-typed variables must be initialized.

var i; // Compile-time error: Implicitly-typed variables must be initialized

i = 100;

Multiple declarations of var variables in a single statement are not allowed.

var i = 100, j = 200, k = 300; // Error: cannot declare var variables in a single statement

//The followings are also valid

var i = 100;

var j = 200;

var k = 300;

var cannot be used for function parameters.

void Display(var param) //Compile-time error

{

Console.Write(param);

}

var can be used in for, and foreach loops.

for(var i = 0; i < 10; i++)

{

Console.WriteLine(i);

}

var can also be used with LINQ queries.

Example: LINQ Query Syntax in C#

// string collection

IList<string> stringList = new List<string>() {

"C# Tutorials",

"VB.NET Tutorials",

"Learn C++",

"MVC Tutorials" ,

"Java"

};

// LINQ Query Syntax

var result = from s in stringList

where s.Contains("Tutorials")

select s;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-AFTZoc)

C# - Data Types

C# is a strongly-typed language. It means we must declare the type of a variable that indicates the kind of values it is going to store, such as integer, float, decimal, text, etc.

The following declares and initialized variables of different data types.

Example: Variables of Different Data Types

string stringVar = "Hello World!!";

int intVar = 100;

float floatVar = 10.2f;

char charVar = 'A';

bool boolVar = true;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-7NK8Om)

C# mainly categorized data types in two types: Value types and Reference types. Value types include simple types (such as int, float, bool, and char), enum types, struct types, and Nullable value types. Reference types include class types, interface types, delegate types, and array types. Learn about [value types and reference types](https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type) in detail in the next chapter.

[](https://www.tutorialsteacher.com/Content/images/csharp/datatypes.png)

Predefined Data Types in C#

C# includes some predefined value types and reference types. The following table lists predefined data types:

| Type | Description | Range | Suffix |
| --- | --- | --- | --- |
| byte | 8-bit unsigned integer | 0 to 255 |  |
| sbyte | 8-bit signed integer | -128 to 127 |  |
| short | 16-bit signed integer | -32,768 to 32,767 |  |
| ushort | 16-bit unsigned integer | 0 to 65,535 |  |
| int | 32-bit signed integer | -2,147,483,648 to 2,147,483,647 |  |
| uint | 32-bit unsigned integer | 0 to 4,294,967,295 | u |
| long | 64-bit signed integer | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | l |
| ulong | 64-bit unsigned integer | 0 to 18,446,744,073,709,551,615 | ul |
| float | 32-bit Single-precision floating point type | -3.402823e38 to 3.402823e38 | f |
| double | 64-bit double-precision floating point type | -1.79769313486232e308 to 1.79769313486232e308 | d |
| decimal | 128-bit decimal type for financial and monetary calculations | (+ or -)1.0 x 10e-28 to 7.9 x 10e28 | m |
| char | 16-bit single Unicode character | Any valid character, e.g. a,\*, \x0058 (hex), or\u0058 (Unicode) |  |
| bool | 8-bit logical true/false value | True or False |  |
| object | Base type of all other types. |  |  |
| string | A sequence of Unicode characters |  |  |
| DateTime | Represents date and time | 0:00:00am 1/1/01 to 11:59:59pm 12/31/9999 |  |

As you can see in the above table that each data type (except string and object) includes value range. The compiler will give an error if the value goes out of datatype's permitted range. For example, int data type's range is -2,147,483,648 to 2,147,483,647. So if you assign a value which is not in this range, then the compiler would give an error.

Example: Compile time error

// compile time error: Cannot implicitly convert type 'long' to 'int'.

int i = 21474836470;

The value of unsigned integers, long, float, double, and decimal type must be suffix by u,l,f,d, and m, respectively.

Example: Value Suffix

uint ui = 100u;

float fl = 10.2f;

long l = 45755452222222l;

ulong ul = 45755452222222ul;

double d = 11452222.555d;

decimal mon = 1000.15m;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-8vDMLs)

Alias vs .NET Type

The predefined data types are alias to their .NET type (CLR class) name. The following table lists alias for predefined data types and related .NET class name.

| Alias | .NET Type | Type |
| --- | --- | --- |
| byte | System.Byte | struct |
| sbyte | System.SByte | struct |
| int | System.Int32 | struct |
| uint | System.UInt32 | struct |
| short | System.Int16 | struct |
| ushort | System.UInt16 | struct |
| long | System.Int64 | struct |
| ulong | System.UInt64 | struct |
| float | System.Single | struct |
| double | System.Double | struct |
| char | System.Char | struct |
| bool | System.Boolean | struct |
| object | System.Object | Class |
| string | System.String | Class |
| decimal | System.Decimal | struct |
| DateTime | System.DateTime | struct |

It means that whether you define a variable of int or Int32, both are the same.

int i = 345;

Int32 i = 345;// same as above

Default Values

Every data type has a default value. Numeric type is 0, boolean has false, and char has '\0' as default value. Use the default(typename) to assign a default value of the data type or C# 7.1 onward, use [default literal](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/default#default-literal).

int i = default(int); // 0

float f = default(float);// 0

decimal d = default(decimal);// 0

bool b = default(bool);// false

char c = default(char);// '\0'

// C# 7.1 onwards

int i = default; // 0

float f = default;// 0

decimal d = default;// 0

bool b = default;// false

char c = default;// '\0'

Conversions

The values of certain data types are automatically converted to different data types in C#. This is called an implicit conversion.

Example: Implicit Conversion

int i = 345;

float f = i;

Console.WriteLine(f); //output: 345

In the above example, the value of an integer variable i is assigned to the variable of float type f because this conversion operation is predefined in C#.

The following is an implicit data type conversion table.

| Implicit Conversion From | To |
| --- | --- |
| sbyte | short, int, long, float, double, decimal |
| byte | short, ushort, int, uint, long, ulong, float, double, decimal |
| short | int, long, float, double, or decimal |
| ushort | int, uint, long, ulong, float, double, or decimal |
| int | long, float, double, or decimal. |
| uint | long, ulong, float, double, or decimal |
| long | float, double, or decimal |
| ulong | float, double, or decimal |
| char | ushort, int, uint, long, ulong, float, double, or decimal |
| float | Double |

Conversions from int, uint, long, or ulong to float and from long or ulong to double may cause a loss of precision. No data type implicitly converted to the char type.

However, not all data types are implicitly converted to other data types. For example, int type cannot be converted to uint implicitly. It must be specified explicitly, as shown below.

Example: Explicit Conversion

public static void Main()

{

int i = 100;

uint u = (uint) i;

Console.Write(i);

}

In the above example, integer i is converted to uint explicitly by specifying uint in the brackets (uint). This will convert an integer to uint.

Numbers in C#

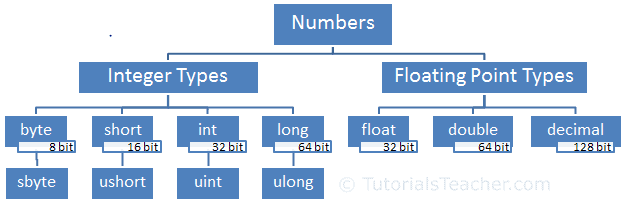
Numbers, in general, can be divided into two types: Integer type and floating-point types.

**Integer type** numbers are whole numbers without decimal points. It can be negative or positive numbers.

**Floating-point type** is numbers with one or more decimal points. It can be negative or positive numbers.

C# includes different data types for integer types and floating-point types based on their size in the memory and capacity to store numbers.

The following figure illustrates numeric types in C#.

[](https://www.tutorialsteacher.com/Content/images/csharp/numbers.PNG)Numeric Types

Integer Types

Integer type numbers are positive or negative whole numbers without decimal points. C# includes four data types for integer numbers: byte, short, int, and long.

Byte

The byte data type stores numbers from 0 to 255. It occupies 8-bit in the memory. The byte keyword is an alias of the [Byte](https://docs.microsoft.com/en-us/dotnet/api/system.byte) struct in .NET.

The sbyte is the same as byte, but it can store negative numbers from -128 to 127. The sbyte keyword is an alias for [SByte](https://docs.microsoft.com/en-us/dotnet/api/system.sbyte) struct in .NET.

Example: byte, sbyte

byte b1 = 255;

byte b2 = -128;// compile-time error: Constant value '-128' cannot be converted to a 'byte'

sbyte sb1 = -128;

sbyte sb2 = 127;

Console.WriteLine(Byte.MaxValue);//255

Console.WriteLine(Byte.MinValue);//0

Console.WriteLine(SByte.MaxValue);//127

Console.WriteLine(SByte.MinValue);//-128

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-LyN0h1)

Short

The short data type is a signed integer that can store numbers from -32,768 to 32,767. It occupies 16-bit memory. The short keyword is an alias for [Int16](https://docs.microsoft.com/en-us/dotnet/api/system.int16) struct in .NET.

The ushort data type is an unsigned integer. It can store only positive numbers from 0 to 65,535. The ushort keyword is an alias for [UInt16](https://docs.microsoft.com/en-us/dotnet/api/system.uint16) struct in .NET.

Example: short, ushort

short s1 = -32768;

short s2 = 32767;

short s3 = 35000;//Compile-time error: Constant value '35000' cannot be converted to a 'short'

ushort us1 = 65535;

ushort us2 = -32000; //Compile-time error: Constant value '-32000' cannot be converted to a 'ushort'

Console.WriteLine(Int16.MaxValue);//32767

Console.WriteLine(Int16.MinValue);//-32768

Console.WriteLine(UInt16.MaxValue);//65535

Console.WriteLine(UInt16.MinValue);//0

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-Wiynwq)

Int

The int data type is 32-bit signed integer. It can store numbers from -2,147,483,648 to 2,147,483,647. The int keyword is an alias of [Int32](https://docs.microsoft.com/en-us/dotnet/api/system.int32) struct in .NET.

The uint is 32-bit unsigned integer. The uint keyword is an alias of [UInt32](https://docs.microsoft.com/en-us/dotnet/api/system.uint32) struct in .NET. It can store positive numbers from 0 to 4,294,967,295. Optionally use U or u suffix after a number to assign it to uint variable.

Example: int, uint

int i = -2147483648;

int j = 2147483647;

int k = 4294967295; //Compile-time error: Cannot implicitly convert type 'uint' to 'int'.

uint ui1 = 4294967295;

uint ui2 =-1; //Compile-time error: Constant value '-1' cannot be converted to a 'uint'

Console.WriteLine(Int32.MaxValue);//2147483647

Console.WriteLine(Int32.MinValue);//-2147483648

Console.WriteLine(UInt32.MaxValue);//4294967295

Console.WriteLine(UInt32.MinValue);//0

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-1Vc0rR)

The int data type is also used for hexadecimal and binary numbers. A hexadecimal number starts with 0x or 0X prefix. C# 7.2 onwards, a binary number starts with 0b or 0B.

Example: Hexadecimal, Binary

int hex = 0x2F;

int binary = 0b\_0010\_1111;

Console.WriteLine(hex);

Console.WriteLine(binary);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-hoET9I)

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Long

The long type is 64-bit signed integers. It can store numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Use l or L suffix with number to assign it to long type variable. The long keyword is an alias of [Int64](https://docs.microsoft.com/en-us/dotnet/api/system.int64) struct in .NET.

The ulong type stores positive numbers from 0 to 18,446,744,073,709,551,615. If a number is suffixed by UL, Ul, uL, ul, LU, Lu, lU, or lu, its type is ulong. The uint keyword is an alias of [UInt64](https://docs.microsoft.com/en-us/dotnet/api/system.uint64) struct in .NET.

Example: long, ulong

long l1 = -9223372036854775808;

long l2 = 9223372036854775807;

ulong ul1 = 18223372036854775808ul;

ulong ul2 = 18223372036854775808UL;

Console.WriteLine(Int64.MaxValue);//9223372036854775807

Console.WriteLine(Int64.MinValue);//-9223372036854775808

Console.WriteLine(UInt64.MaxValue);//18446744073709551615

Console.WriteLine(UInt64.MinValue);//0

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-E7QBdf)

Floating Point Types

Floating-point numbers are positive or negative numbers with one or more decimal points. C# includes three data types for floating-point numbers: float, double, and decimal.

Float

The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. It occupies 4 bytes in the memory. The float keyword is an alias of [Single](https://docs.microsoft.com/en-us/dotnet/api/system.single) struct in .NET.

Use f or F suffix with literal to make it float type.

Example: float

float f1 = 123456.5F;

float f2 = 1.123456f;

Console.WriteLine(f1);//123456.5

Console.WriteLine(f2);//1.123456

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-AoQSDL)

Double

The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. It occupies 8 bytes in the memory. The double keyword is an alias of the [Double](https://docs.microsoft.com/en-us/dotnet/api/system.double) struct in .NET.

Use d or D suffix with literal to make it double type.

Example: double

double d1 = 12345678912345.5d;

double d2 = 1.123456789123456d;

Console.WriteLine(d1);//12345678912345.5

Console.WriteLine(d2);//1.123456789123456

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-0IVbLA)

Decimal

The decimal data type can store fractional numbers from ±1.0 x 10-28 to ±7.9228 x 1028. It occupies 16 bytes in the memory. The decimal is a keyword alias of the [Decimal](https://docs.microsoft.com/en-us/dotnet/api/system.decimal) struct in .NET.

The decimal type has more precision and a smaller range than both float and double, and so it is appropriate for financial and monetary calculations.

Use m or M suffix with literal to make it decimal type.

Example: decimal

decimal d1 = 123456789123456789123456789.5m;

decimal d2 = 1.1234567891345679123456789123m;

Console.WriteLine(d1);

Console.WriteLine(d2);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-bU1v7O)

Scientific Notation

Use e or E to indicate the power of 10 as exponent part of scientific notation with float, double or decimal.

Example:

double d = 0.12e2;

Console.WriteLine(d); // 12;

float f = 123.45e-2f;

Console.WriteLine(f); // 1.2345

decimal m = 1.2e6m;

Console.WriteLine(m);// 1200000

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ieZZpB)

C# Strings

In C#, a string is a series of characters that is used to represent text. It can be a character, a word or a long passage surrounded with the double quotes ". The following are string literals.

Example: String Literals

"S"

"String"

"This is a string."

C# provides the String data type to store string literals. A variable of the string type can be declared and assign string literal, as shown below.

Example: String Type Variables

string ch = "S";

string word = "String";

string text = "This is a string.";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-EVNe9D)

The maximum size of a String object in memory is 2GB or about 1 billion characters. However, practically it will be less depending upon CPU and memory of the computer.

There two ways to declare a string variable in C#. Using System.String class and using string keyword. Both are the same and make no difference. Learn [string vs String](https://www.tutorialsteacher.com/articles/difference-between-string-and-string-in-csharp) for more info.

Example: String and string

string str1 = "Hello"; // uses string keyword

String str2 = "Hello"; // uses System.String class

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-RN0bzx)

In C#, a string is a collection or an array of characters. So, string can be created using a char array or accessed like a char array.

Example: String as char Array

char[] chars = {'H','e','l','l','o'};

string str1 = new string(chars);

String str2 = new String(chars);

foreach (char c in str1)

{

Console.WriteLine(c);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-rV9zMO)

Special Characters

A text in the real world can include any character. In C#, because a string is surrounded with double quotes, it cannot include " in a string. The following will give a compile-time error.

Example: Invalid String

string text = "This is a "string" in C#.";

C# includes escaping character \ (backslash) before these special characters to include in a string.

Use backslash \ before double quotes and some special characters such as \,\n,\r,\t, etc. to include it in a string.

Example: Escape Char \

string text = "This is a \"string\" in C#.";

string str = "xyzdef\\rabc";

string path = "\\\\mypc\\ shared\\project";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-gfNWt9)

Verbatim Strings

It is tedious to prefix \ to include every special characters. Verbatim string in C# allows a special characters and line brakes. Verbatim string can be created by prefixing @ symbol before double quotes.

Example: Escape Sequence

string str = @"xyzdef\rabc";

string path = @"\\mypc\shared\project";

string email = @"test@test.com";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GVVZRQ)

The @ symbol can also be used to declare a multi-line string.

Example: Multi-line String

string str1 = "this is a \n" +

"multi line \n" +

"string";

// Verbatim string

string str2 = @"this is a

multi line

string";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ITEdig)

Please note that you cannot use a backslash to allow " in a varbatim string. If you wish to include @, then use double double-quotes "" to include " in a verbatim string.

string text = "This is a \"string\" in C#."; // valid

string text = @"This is a "string." in C#."; // error

string text = @"This is a \"string\" in C#."; // error

string text = @"This is a ""string"" in C#."; // valid

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

String Concatenation

Multiple strings can be concatenated with + operator.

Example: String Concatenation

string name = "Mr." + "James " + "Bond" + ", Code: 007";

string firstName = "James";

string lastName = "Bond";

string code = "007";

string agent = "Mr." + firstName + " " + lastName + ", Code: " + code;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-vLTRSV)

A String is immutable in C#. It means it is read-only and cannot be changed once created in the memory. Each time you concatenate strings, .NET CLR will create a new memory location for the concatenated string. So, it is recommended to use [StringBuilder](https://www.tutorialsteacher.com/csharp/csharp-stringbuilder) instead of string if you concatenate more than five strings.

String Interpolation

String interpolation is a better way of concatenating strings. We use + sign to concatenate string variables with static strings.

C# 6 includes a special character $ to identify an interpolated string. An interpolated string is a mixture of static string and string variable where string variables should be in {} brackets.

Example: String Interpolation

string firstName = "James";

string lastName = "Bond";

string code = "007";

string fullName = $"Mr. {firstName} {lastName}, Code: {code}";

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-APwAWB)

In the above example of interpolation, $ indicates the interpolated string, and {} includes string variable to be incorporated with a string.

Use two braces, "{{" or "}}" to include { or } in a string.

Working with Date and Time in C#

C# includes DateTime struct to work with dates and times.

To work with date and time in C#, create an object of the DateTime struct using the new keyword. The following creates a DateTime object with the default value.

Example: Create DateTime Object

DateTime dt = new DateTime(); // assigns default value 01/01/0001 00:00:00

The default and the lowest value of a DateTime object is January 1, 0001 00:00:00 (midnight). The maximum value can be December 31, 9999 11:59:59 P.M.

Use different constructors of the DateTime struct to assign an initial value to a DateTime object.

Example: Set Date & Time

//assigns default value 01/01/0001 00:00:00

DateTime dt1 = new DateTime();

//assigns year, month, day

DateTime dt2 = new DateTime(2015, 12, 31);

//assigns year, month, day, hour, min, seconds

DateTime dt3 = new DateTime(2015, 12, 31, 5, 10, 20);

//assigns year, month, day, hour, min, seconds, UTC timezone

DateTime dt4 = new DateTime(2015, 12, 31, 5, 10, 20, DateTimeKind.Utc);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-oousl5)

In the above example, we specified a year, a month, and a day in the constructor. The year can be from 0001 to 9999, and the Month can be from 1 to 12, and the day can be from 1 to 31. Setting any other value out of these ranges will result in a run-time exception.

Example: Invalid Date

DateTime dt = new DateTime(2015, 12, 32); //throws exception: day out of range

Use different [DateTime constructors](https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=netframework-4.8#constructors) to set date, time, time zone, calendar, and culture.

Ticks

Ticks is a date and time expressed in the number of 100-nanosecond intervals that have elapsed since January 1, 0001, at 00:00:00.000 in the Gregorian calendar. The following initializes a DateTime object with the number of ticks.

Example: Ticks

DateTime dt = new DateTime(636370000000000000);

DateTime.MinValue.Ticks; //min value of ticks

DateTime.MaxValue.Ticks; // max value of ticks

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-IoBmg8)

DateTime Static Fields

The DateTime struct includes static fields, properties, and methods. The following example demonstrates important static fields and properties.

Example: Static Fields

DateTime currentDateTime = DateTime.Now; //returns current date and time

DateTime todaysDate = DateTime.Today; // returns today's date

DateTime currentDateTimeUTC = DateTime.UtcNow;// returns current UTC date and time

DateTime maxDateTimeValue = DateTime.MaxValue; // returns max value of DateTime

DateTime minDateTimeValue = DateTime.MinValue; // returns min value of DateTime

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-acYj0c)

TimeSpan

TimeSpan is a struct that is used to represent time in days, hour, minutes, seconds, and milliseconds.

Example: TimeSpan

DateTime dt = new DateTime(2015, 12, 31);

TimeSpan ts = new TimeSpan(25,20,55);

DateTime newDate = dt.Add(ts);

Console.WriteLine(newDate);//1/1/2016 1:20:55 AM

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-rtRFYd)

Subtraction of two dates results in TimeSpan.

Example: Subtract Dates

DateTime dt1 = new DateTime(2015, 12, 31);

DateTime dt2 = new DateTime(2016, 2, 2);

TimeSpan result = dt2.Subtract(dt1);//33.00:00:00

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GTHzgf)

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Operators

The DateTime struct overloads +, -, ==, !=, >, <, <=, >= operators to ease out addition, subtraction, and comparison of dates. These make it easy to work with dates.

Example: Operators

DateTime dt1 = new DateTime(2015, 12, 20);

DateTime dt2 = new DateTime(2016, 12, 31, 5, 10, 20);

TimeSpan time = new TimeSpan(10, 5, 25, 50);

Console.WriteLine(dt2 + time); // 1/10/2017 10:36:10 AM

Console.WriteLine(dt2 - dt1); //377.05:10:20

Console.WriteLine(dt1 == dt2); //False

Console.WriteLine(dt1 != dt2); //True

Console.WriteLine(dt1 > dt2); //False

Console.WriteLine(dt1 < dt2); //True

Console.WriteLine(dt1 >= dt2); //False

Console.WriteLine(dt1 <= dt2);//True

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-uWFGbZ)

Convert String to DateTime

A valid date and time string can be converted to a DateTime object using [Parse()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.parse?view=netframework-4.8), [ParseExact()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.parseexact?view=netframework-4.8), [TryParse()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tryparse?view=netframework-4.8) and [TryParseExact()](https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tryparseexact?view=netframework-4.8) methods.

The Parse() and ParseExact() methods will throw an exception if the specified string is not a valid representation of a date and time. So, it's recommended to use TryParse() or TryParseExact() method because they return false if a string is not valid.

Example:

var str = "5/12/2020";

DateTime dt;

var isValidDate = DateTime.TryParse(str, out dt);

if(isValidDate)

Console.WriteLine(dt);

else

Console.WriteLine($"{str} is not a valid date string");

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-2frLly)

C# Enumerations Type - Enum

In C#, an enum (or enumeration type) is used to assign constant names to a group of numeric integer values. It makes constant values more readable, for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.

An enum is defined using the enum keyword, directly inside a namespace, class, or structure. All the constant names can be declared inside the curly brackets and separated by a comma. The following defines an enum for the weekdays.

Example: Define an Enum

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Above, the WeekDays enum declares members in each line separated by a comma.

Enum Values

If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default. The first member of an enum will be 0, and the value of each successive enum member is increased by 1.

Example: Default Enum Values

enum WeekDays

{

Monday, // 0

Tuesday, // 1

Wednesday, // 2

Thursday, // 3

Friday, // 4

Saturday, // 5

Sunday // 6

}

You can assign different values to enum member. A change in the default value of an enum member will automatically assign incremental values to the other members sequentially.

Example: Assign Values to Enum Members

enum Categories

{

Electronics, // 0

Food, // 1

Automotive = 6, // 6

Arts, // 7

BeautyCare, // 8

Fashion // 9

}

You can even assign different values to each member.

Example: Assign Values to Enum Members

enum Categories

{

Electronics = 1,

Food = 5,

Automotive = 6,

Arts = 10,

BeautyCare = 11,

Fashion = 15,

WomanFashion = 15

}

The enum can be of any numeric data type such as byte, sbyte, short, ushort, int, uint, long, or ulong. However, an enum cannot be a string type.

Specify the type after enum name as : type. The following defines the byte enum.

Example: byte Enum

enum Categories: byte

{

Electronics = 1,

Food = 5,

Automotive = 6,

Arts = 10,

BeautyCare = 11,

Fashion = 15

}

Access an Enum

An enum can be accessed using the dot syntax: enum.member

Example: Access Enum

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(WeekDays.Monday); // Monday

Console.WriteLine(WeekDays.Tuesday); // Tuesday

Console.WriteLine(WeekDays.Wednesday); // Wednesday

Console.WriteLine(WeekDays.Thursday); // Thursday

Console.WriteLine(WeekDays.Friday); // Friday

Console.WriteLine(WeekDays.Saturday); // Saturday

Console.WriteLine(WeekDays.Sunday); // Sunday

Conversion

Explicit casting is required to convert from an enum type to its underlying integral type.

Example: Enum Conversion

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(WeekDays.Friday); //output: Friday

int day = (int) WeekDays.Friday; // enum to int conversion

Console.WriteLine(day); //output: 4

var wd = (WeekDays) 5; // int to enum conversion

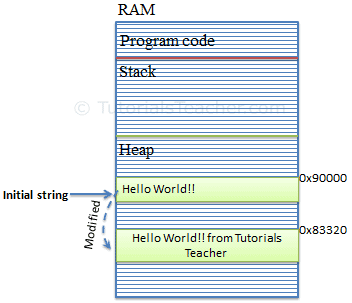
Console.WriteLine(wd);//output: Saturday

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-mk8Ojx)

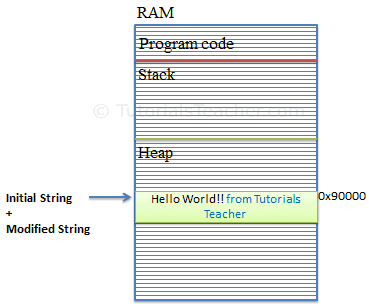
C# - StringBuilder

Updated on: June 26, 2020

In C#, the string type is immutable. It means a string cannot be changed once created. For example, a new string, "Hello World!" will occupy a memory space on the heap. Now, by changing the initial string "Hello World!" to "Hello World! from Tutorials Teacher" will create a new string object on the memory heap instead of modifying an original string at the same memory address. This behavior would hinder the performance if the original string changed multiple times by replacing, appending, removing, or inserting new strings in the original string.

[](https://www.tutorialsteacher.com/Content/images/csharp/string-memory.png)Memory Allocation for String Object

To solve this problem, C# introduced the StringBuilder in the [System.Text](https://docs.microsoft.com/en-us/dotnet/api/system.text) namespace. The StringBuilder doesn't create a new object in the memory but dynamically expands memory to accommodate the modified string.

[](https://www.tutorialsteacher.com/Content/images/csharp/stringbuilder-memory.png)Memory Allocation for StringBuilder Object

Creating a StringBuilder Object

You can create an object of the StringBuilder class using the new keyword and passing an initial string. The following example demonstrates creating StringBuilder objects.

Example: StringBuilder

using System.Text; // include at the top

StringBuilder sb = new StringBuilder(); //string will be appended later

//or

StringBuilder sb = new StringBuilder("Hello World!");

Optionally, you can also specify the maximum capacity of the StringBuilder object using overloaded constructors, as shown below.

Example: StringBuilder

StringBuilder sb = new StringBuilder(50); //string will be appended later

//or

StringBuilder sb = new StringBuilder("Hello World!", 50);

Above, C# allocates a maximum of 50 spaces sequentially on the memory heap. This capacity will automatically be doubled once it reaches the specified capacity. You can also use the capacity or length property to set or retrieve the StringBuilder object's capacity.

You can iterate the using [for loop](https://www.tutorialsteacher.com/csharp/csharp-for-loop) to get or set a character at the specified index.

Example: StringBuilder Iteration

StringBuilder sb = new StringBuilder("Hello World!");

for(int i = 0; i < sb.Length; i++)

Console.Write(sb[i]); // output: Hello World!

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-AxUF7x)

Retrieve String from StringBuilder

The StringBuilder is not the string. Use the ToString() method to retrieve a string from the StringBuilder object.

Example: Retrieve String from StringBuilder

StringBuilder sb = new StringBuilder("Hello World!");

var greet = sb.ToString(); //returns "Hello World!"

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Add/Append String to StringBuilder

Use the Append() method to append a string at the end of the current StringBuilder object. If a StringBuilder does not contain any string yet, it will add it. The AppendLine() method append a string with the newline character at the end.

Example: Adding or Appending Strings in StringBuilder

StringBuilder sb = new StringBuilder();

sb.Append("Hello ");

sb.AppendLine("World!");

sb.AppendLine("Hello C#");

Console.WriteLine(sb);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-dxwlys)

Output:

Hello World!  
Hello C#.

Append Formated String to StringBuilder

Use the AppendFormat() method to format an input string into the specified format and append it.

Example: AppendFormat()

StringBuilder sbAmout = new StringBuilder("Your total amount is ");

sbAmout.AppendFormat("{0:C} ", 25);

Console.WriteLine(sbAmout);//output: Your total amount is $ 25.00

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-5fVzrl)

Insert String into StringBuilder

Use the Insert() method inserts a string at the specified index in the StringBuilder object.

Example: Insert()

StringBuilder sb = new StringBuilder("Hello World!");

sb.Insert(5," C#");

Console.WriteLine(sb); //output: Hello C# World!

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-y5uoiJ)

Remove String in StringBuilder

Use the Remove() method to remove a string from the specified index and up to the specified length.

Example: Remove()

StringBuilder sb = new StringBuilder("Hello World!",50);

sb.Remove(6, 7);

Console.WriteLine(sb); //output: Hello

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-C8kClJ)

Replace String in StringBuilder

Use the Replace() method to replace all the specified string occurrences with the specified replacement string.

Example: Replace()

StringBuilder sb = new StringBuilder("Hello World!");

sb.Replace("World", "C#");

Console.WriteLine(sb);//output: Hello C#!

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-xONjOl)

https://www.tutorialsteacher.com/Content/images/bulb-glow.png Points to Remember :

1. StringBuilder is mutable.
2. StringBuilder performs faster than string when appending multiple string values.
3. Use StringBuilder when you need to append more than three or four strings.
4. Use the Append() method to add or append strings to the StringBuilder object.
5. Use the ToString() method to retrieve a string from the StringBuilder object.

C# - Anonymous Type

Updated on: May 2, 2020

In C#, an anonymous type is a type (class) without any name that can contain public read-only properties only. It cannot contain other members, such as fields, methods, events, etc.

You create an anonymous type using the *new* operator with an [object initializer](https://www.tutorialsteacher.com/csharp/csharp-object-initializer) syntax. The [implicitly typed variable- var](https://www.tutorialsteacher.com/csharp/csharp-var-implicit-typed-local-variable) is used to hold the reference of anonymous types.

The following example demonstrates creating an anonymous type variable student that contains three properties named Id, FirstName, and LastName.

Example: Anonymous Type

var student = new { Id = 1, FirstName = "James", LastName = "Bond" };

The properties of anonymous types are read-only and cannot be initialized with a null, anonymous function, or a pointer type. The properties can be accessed using dot (.) notation, same as object properties. However, you cannot change the values of properties as they are read-only.

Example: Access Anonymous Type

var student = new { Id = 1, FirstName = "James", LastName = "Bond" };

Console.WriteLine(student.Id); //output: 1

Console.WriteLine(student.FirstName); //output: James

Console.WriteLine(student.LastName); //output: Bond

student.Id = 2;//Error: cannot chage value

student.FirstName = "Steve";//Error: cannot chage value

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-hkLMr2)

An anonymous type's property can include another anonymous type.

Example: Nested Anonymous Type

var student = new {

Id = 1,

FirstName = "James",

LastName = "Bond",

Address = new { Id = 1, City = "London", Country = "UK" }

};

You can create an array of anonymous types also.

Example: Array of Anonymous Types

var students = new[] {

new { Id = 1, FirstName = "James", LastName = "Bond" },

new { Id = 2, FirstName = "Steve", LastName = "Jobs" },

new { Id = 3, FirstName = "Bill", LastName = "Gates" }

};

An anonymous type will always be local to the method where it is defined. It cannot be returned from the method. However, an anonymous type can be passed to the method as object type parameter, but it is not recommended. If you need to pass it to another method, then use struct or class instead of an anonymous type.

Mostly, anonymous types are created using the [Select](https://www.tutorialsteacher.com/linq/linq-projection-operators) clause of a LINQ queries to return a subset of the properties from each object in the collection.

Example: LINQ Query returns an Anonymous Type

class Program

{

static void Main(string[] args)

{

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", age = 18 },

new Student() { StudentID = 2, StudentName = "Steve", age = 21 },

new Student() { StudentID = 3, StudentName = "Bill", age = 18 },

new Student() { StudentID = 4, StudentName = "Ram" , age = 20 },

new Student() { StudentID = 5, StudentName = "Ron" , age = 21 }

};

var students = from s in studentList

select new { Id = s.StudentID, Name = s.StudentName };

foreach(var stud in students)

Console.WriteLine(stud.Id + "-" + stud.Name);

}

}

public class Student

{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int age { get; set; }

}

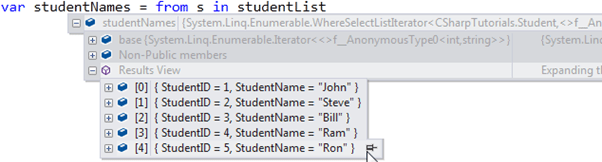
[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-igsNNA)

Output:

1-John  
2-Steve  
3-Bill  
4-Ram  
5-Ron

In the above example, a select clause in the LINQ query selects only StudentID and StudentName properties and renames it to Id and Name, respectively. Thus, it is useful in saving memory and unnecessary code. The query result collection includes only StudentID and StudentName properties, as shown in the following debug view.

Visual Studio supports IntelliSense for anonymous types, as shown below.

[](https://www.tutorialsteacher.com/Content/images/csharp/anonymoustype-debugview.png)Anonymous Type Intellisense Support in Visual Studio

Internally, all the anonymous types are directly derived from the System.Object class. The compiler generates a class with some auto-generated name and applies the appropriate type to each property based on the value expression. Although your code cannot access it. Use GetType() method to see the name.

Example: Internal Name of an Anonymous Type

static void Main(string[] args)

{

var student = new { Id = 1, FirstName = "James", LastName = "Bond" };

Console.WriteLine(student.GetType().ToString());

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-8ibfMq)

C# - Dynamic Types

C# 4.0 (.NET 4.5) introduced a new type called dynamic that avoids compile-time type checking. A dynamic type escapes type checking at compile-time; instead, it resolves type at run time.

A dynamic type variables are defined using the dynamic keyword.

Example: dynamic Variable

dynamic MyDynamicVar = 1;

The compiler compiles dynamic types into object types in most cases. However, the actual type of a dynamic type variable would be resolved at run-time.

Example: dynamic Type at run-time

dynamic MyDynamicVar = 1;

Console.WriteLine(MyDynamicVar.GetType());

Output:

System.Int32

Dynamic types change types at run-time based on the assigned value. The following example shows how a dynamic variable changes type based on assigned value.

Example: dynamic

static void Main(string[] args)

{

dynamic MyDynamicVar = 100;

Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

MyDynamicVar = "Hello World!!";

Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

MyDynamicVar = true;

Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

MyDynamicVar = DateTime.Now;

Console.WriteLine("Value: {0}, Type: {1}", MyDynamicVar, MyDynamicVar.GetType());

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-L98yXw)

Output:

Value: 100, Type: System.Int32  
Value: Hello World!!, Type: System.String  
Value: True, Type: System.Boolean  
Value: 01-01-2014, Type: System.DateTime

The dynamic type variables is converted to other types implicitly.

Example: dynamic Type Conversion

dynamic d1 = 100;

int i = d1;

d1 = "Hello";

string greet = d1;

d1 = DateTime.Now;

DateTime dt = d1;

Methods and Parameters

If you assign a class object to the dynamic type, then the compiler would not check for correct methods and properties name of a dynamic type that holds the custom class object. Consider the following example.

Example: Calling Methods

class Program

{

static void Main(string[] args)

{

dynamic stud = new Student();

stud.DisplayStudentInfo(1, "Bill");// run-time error, no compile-time error

stud.DisplayStudentInfo("1");// run-time error, no compile-time error

stud.FakeMethod();// run-time error, no compile-time error

}

}

public class Student

{

public void DisplayStudentInfo(int id)

{

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-A7jGc0)

In the above example, the C# compiler does not check for the number of parameters, parameters type, or non-existent. It validates these things at run-time, and if it is not valid, then throws a run-time exception. Note that Visual Studio IntelliSense is not supported for the dynamic types. Note that Visual Studio IntelliSense is not supported for the dynamic types.

The dynamic language runtime (DLR) API provides the infrastructure that supports the dynamic type in C#.

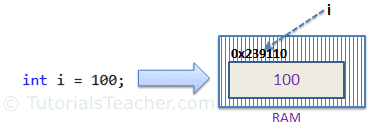
Value Type

A data type is a value type if it holds a data value within its own memory space. It means the variables of these data types directly contain values.

tipAll the value types derive from *System.ValueType*, which in-turn, derives from *System.Object*.

For example, consider integer variable int i = 100;

The system stores 100 in the memory space allocated for the variable i. The following image illustrates how 100 is stored at some hypothetical location in the memory (0x239110) for 'i':

[](https://www.tutorialsteacher.com/Content/images/csharp/value-type-memory-allocation.png)Memory Allocation of Value Type Variable

The following data types are all of value type:

* bool
* byte
* char
* decimal
* double
* enum
* float
* int
* long
* sbyte
* short
* struct
* uint
* ulong
* ushort

Passing Value Type Variables

When you pass a value-type variable from one method to another, the system creates a separate copy of a variable in another method. If value got changed in the one method, it wouldn't affect the variable in another method.

Example: Passing Value Type Variables

static void ChangeValue(int x)

{

x = 200;

Console.WriteLine(x);

}

static void Main(string[] args)

{

int i = 100;

Console.WriteLine(i);

ChangeValue(i);

Console.WriteLine(i);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ba76t1)

Output:

100  
200  
100

In the above example, variable i in the Main() method remains unchanged even after we pass it to the ChangeValue() method and change it's value there.

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

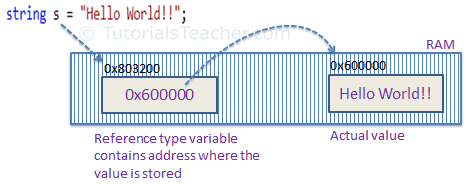
Reference Type

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

For example, consider the following string variable:

string s = "Hello World!!";

The following image shows how the system allocates the memory for the above string variable.

[](https://www.tutorialsteacher.com/Content/images/csharp/raference-type-memory-allocation.png)Memory Allocation of Reference Type Variable

As you can see in the above image, the system selects a random location in memory (0x803200) for the variable s. The value of a variable s is 0x600000, which is the memory address of the actual data value. Thus, reference type stores the address of the location where the actual value is stored instead of the value itself.

The followings are reference type data types:

* String
* Arrays (even if their elements are value types)
* Class
* Delegate

Passing Reference Type Variables

When you pass a reference type variable from one method to another, it doesn't create a new copy; instead, it passes the variable's address. So, If we change the value of a variable in a method, it will also be reflected in the calling method.

Example: Passing Reference Type Variable

static void ChangeReferenceType(Student std2)

{

std2.StudentName = "Steve";

}

static void Main(string[] args)

{

Student std1 = new Student();

std1.StudentName = "Bill";

ChangeReferenceType(std1);

Console.WriteLine(std1.StudentName);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-WL8Xgk)

Output:

Steve

In the above example, we pass the Student object std1 to the ChangeReferenceType() method. Here, it actually pass the memory address of std1. Thus, when the ChangeReferenceType() method changes StudentName, it is actually changing StudentName of std1 object, because std1 and std2 are both pointing to the same address in memory.

[String](https://www.tutorialsteacher.com/csharp/csharp-string) is a reference type, but it is immutable. It means once we assigned a value, it cannot be changed. If we change a string value, then the compiler creates a new string object in the memory and point a variable to the new memory location. So, passing a string value to a function will create a new variable in the memory, and any change in the value in the function will not be reflected in the original value, as shown below.

Example: Passing String

static void ChangeReferenceType(string name)

{

name = "Steve";

}

static void Main(string[] args)

{

string name = "Bill";

ChangeReferenceType(name);

Console.WriteLine(name);

}

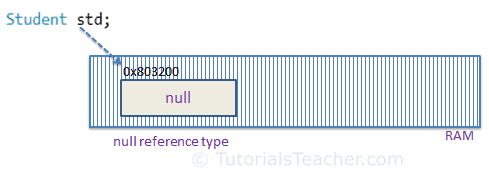
[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-g2FG7h)

Output:

Bill

Null

The default value of a reference type variable is null when they are not initialized. Null means not refering to any object.

[](https://www.tutorialsteacher.com/Content/images/csharp/null.png)Null Reference Type

A value type variable cannot be null because it holds value, not a memory address. C# 2.0 introduced [nullable types](https://www.tutorialsteacher.com/csharp/csharp-nullable-types), using which you can assign null to a value type variable or declare a value type variable without assigning a value to it.

C# - Interface

In the human world, a contract between the two or more humans binds them to act as per the contract. In the same way, an interface includes the declarations of related functionalities. The entities that implement the interface must provide the implementation of declared functionalities.

In C#, an interface can be defined using the interface keyword. An interface can contain declarations of methods, properties, indexers, and events. However, it cannot contain fields, auto-implemented properties.

The following interface declares some basic functionalities for the file operations.

Example: C# Interface

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

You cannot apply access modifiers to interface members. All the members are public by default. If you use an access modifier in an interface, then the C# compiler will give a compile-time error "The modifier 'public/private/protected' is not valid for this item.". (Visual Studio will show an error immediately without compilation.)

Example: Invalid Interface with Access Modifiers

interface IFile

{

protected void ReadFile(); //compile-time error

private void WriteFile(string text);//compile-time error

}

An interface can only contain declarations but not implementations. The following will give a compile-time error.

Example: Invalid Interface with Implementation

interface IFile

{

void ReadFile();

void WriteFile(string text){

Console.Write(text); //error: cannot implement method

}

}

Implementing an Interface

A class or a Struct can implement one or more interfaces using colon (:).

Syntax: <Class or Struct Name> : <Interface Name>

For example, the following class implements the IFile interface implicitly.

Example: Interface Implementation

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

class FileInfo : IFile

{

public void ReadFile()

{

Console.WriteLine("Reading File");

}

public void WriteFile(string text)

{

Console.WriteLine("Writing to file");

}

}

In the above example, the FileInfo class implements the IFile interface. It defines all the members of the IFile interface with public access modifier. The FileInfo class can also contain members other than interface members.

 Note:

Interface members must be implemented with the public modifier; otherwise, the compiler will give compile-time errors.

You can create an object of the class and assign it to a variable of an interface type, as shown below.

Example: Interface Implementation

public class Program

{

public static void Main()

{

IFile file1 = new FileInfo();

FileInfo file2 = new FileInfo();

file1.ReadFile();

file1.WriteFile("content");

file2.ReadFile();

file2.WriteFile("content");

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-NKawDI)

Above, we created objects of the FileInfo class and assign it to IFile type variable and FileInfo type variable. When interface implemented implicitly, you can access IFile members with the IFile type variables as well as FileInfo type variable.

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Explicit Implementation

An interface can be implemented explicitly using <InterfaceName>.<MemberName>. Explicit implementation is useful when class is implementing multiple interfaces; thereby, it is more readable and eliminates the confusion. It is also useful if interfaces have the same method name coincidently.

 Note:

Do not use *public* modifier with an explicit implementation. It will give a compile-time error.

Example: Explicit Implementation

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

class FileInfo : IFile

{

void IFile.ReadFile()

{

Console.WriteLine("Reading File");

}

void IFile.WriteFile(string text)

{

Console.WriteLine("Writing to file");

}

}

When you implement an interface explicitly, you can access interface members only through the instance of an interface type.

Example: Explicit Implementation

interface IFile

{

void ReadFile();

void WriteFile(string text);

}

class FileInfo : IFile

{

void IFile.ReadFile()

{

Console.WriteLine("Reading File");

}

void IFile.WriteFile(string text)

{

Console.WriteLine("Writing to file");

}

public void Search(string text)

{

Console.WriteLine("Searching in file");

}

}

public class Program

{

public static void Main()

{

IFile file1 = new FileInfo();

FileInfo file2 = new FileInfo();

file1.ReadFile();

file1.WriteFile("content");

//file1.Search("text to be searched")//compile-time error

file2.Search("text to be searched");

//file2.ReadFile(); //compile-time error

//file2.WriteFile("content"); //compile-time error

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-2b5QYO)

In the above example, file1 object can only access members of IFile, and file2 can only access members of FileInfo class. This is the limitation of explicit implementation.

Implementing Multiple Interfaces

A class or struct can implement multiple interfaces. It must provide the implementation of all the members of all interfaces.

Example: Implement Multiple Interfaces

interface IFile

{

void ReadFile();

}

interface IBinaryFile

{

void OpenBinaryFile();

void ReadFile();

}

class FileInfo : IFile, IBinaryFile

{

void IFile.ReadFile()

{

Console.WriteLine("Reading Text File");

}

void IBinaryFile.OpenBinaryFile()

{

Console.WriteLine("Opening Binary File");

}

void IBinaryFile.ReadFile()

{

Console.WriteLine("Reading Binary File");

}

public void Search(string text)

{

Console.WriteLine("Searching in File");

}

}

public class Program

{

public static void Main()

{

IFile file1 = new FileInfo();

IBinaryFile file2 = new FileInfo();

FileInfo file3 = new FileInfo();

file1.ReadFile();

//file1.OpenBinaryFile(); //compile-time error

//file1.SearchFile("text to be searched"); //compile-time error

file2.OpenBinaryFile();

file2.ReadFile();

//file2.SearchFile("text to be searched"); //compile-time error

file3.Search("text to be searched");

//file3.ReadFile(); //compile-time error

//file3.OpenBinaryFile(); //compile-time error

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-nxCU4v)

Above, the FileInfo implements two interfaces IFile and IBinaryFile explicitly. It is recommended to implement interfaces explicitly when implementing multiple interfaces to avoid confusion and more readability.

https://www.tutorialsteacher.com/Content/images/bulb-glow.png Points to Remember :

1. Interface can contain declarations of method, properties, indexers, and events.
2. Interface cannot include private, protected, or internal members. All the members are public by default.
3. Interface cannot contain fields, and auto-implemented properties.
4. A class or a struct can implement one or more interfaces implicitly or explicitly. Use public modifier when implementing interface implicitly, whereas don't use it in case of explicit implementation.
5. Implement interface explicitly using InterfaceName.MemberName.
6. An interface can inherit one or more interfaces.

C# Operators

Operators in C# are some special symbols that perform some action on operands. In mathematics, the plus symbol (+) do the sum of the left and right numbers. In the same way, C# includes various operators for different types of operations.

The following example demonstrates the + operator in C#.

Example: + Operator

int x = 5 + 5;

int y = 10 + x;

int z = x + y;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-MjxOfI)

In the above example, + operator adds two number literals and assign the result to a variable. It also adds the values of two int variables and assigns the result to a variable.

Some operators behave differently based on the type of the operands. For example, + operator concatenates two strings.

Example: + Operator with Strings

string greet1 = "Hello " + "World!";

string greet2 = greeting + name;

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-B1aOTu)

 Note:

There are two types of operators in C#, Unary operators and Binary operators. Unary operators act on single operand, whereas binary operators act on two operands (left-hand side and right-hand side operand of an operator).

C# includes the following categories of operators:

* [Arithmetic operators](https://www.tutorialsteacher.com/csharp/csharp-operators#arithmetic-operators)
* [Assignment operators](https://www.tutorialsteacher.com/csharp/csharp-operators#assignment-operators)
* [Comparison operators](https://www.tutorialsteacher.com/csharp/csharp-operators#comparison-operators)
* [Equality operators](https://www.tutorialsteacher.com/csharp/csharp-operators#equality-operators)
* [Boolean logical operators](https://www.tutorialsteacher.com/csharp/csharp-operators#logical-operators)
* [Betwise and shift operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators)
* [Member access operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/member-access-operators)
* [Type-cast operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast)
* [Pointer related operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/pointer-related-operators)

Arithmetic Operators

The arithmetic operators perform arithmetic operations on all the numeric type operands such as sbyte, byte, short, ushort, int, uint, long, ulong, float, double, and decimal.

| Operator | Name | Description | Example |  |
| --- | --- | --- | --- | --- |
| + | Addition | Computes the sum of left and right operands. | int x = 5 + 5; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-AQ5t4m) |
| - | Subtraction | Subtract the right operand from the left operand | int x = 5 - 1; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-Tz6rEA) |
| \* | Multiplication | Multiply left and right operand | int x = 5 \* 1; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-2xKw5p) |
| / | Division | Divides the left operand by the right operand | int x = 10 / 2; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-278Kl1) |
| % | Reminder | Computes the remainder after dividing its left operand by its right operand | int x = 5 % 2; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-HzWx8H) |
| ++ | Unary increment | Unary increment ++ operator increases its operand by 1 | x++ | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-2GrYMx) |
| -- | Unary decrement | Unary decrement -- operator decreases its operand by 1 | x-- | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-sGd0gG) |
| + | Unary plus | Returns the value of operand | +5 | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ljO0lc) |
| - | Unary minus | Computes the numeric negation of its operand. | -5 | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ljO0lc) |

Assignment Operators

The assignment operator = assigns its right had value to its left-hand variable, property, or indexer. It can also be used with other arithmetic, Boolean logical, and bitwise operators.

| Operator | Name | Description | Example |  |
| --- | --- | --- | --- | --- |
| = | Assignment | Assigns its right had value to its left-hand variable, property or indexer. | x = 10; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-OnvjFQ) |
| x op= y | Compound assignment | Short form of x =x op y where op = any arithmetic, Boolean logical, and bitwise operator. | x += 5; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ZYXCLS) |
| ??= | Null-coalescing assignment | C# 8 onwards, ??= assigns value of the right operand only if the left operand is null | x ??= 5; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-fbJUvB) |

Comparison Operators

Comparison operators compre two numeric operands and returns true or false.

| Operator | Description | Example |  |
| --- | --- | --- | --- |
| < | Returns true if the right operand is less than the left operand | x < y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-bNAEm6) |
| > | Returns true if the right operand is greater than the left operand | x > y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-AR6gaP) |
| <= | Returns true if the right operand is less than or equal to the left operand | x <= y | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-QItokj) |
| >= | Returns true if the right operand is greater than or equal to the left operand | x >= y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-wyYRPu) |

Microsoft To Launch Cross-Platform Email Client Codenamed Project Monarch

Equality Operators

The equality operator checks whether the two operands are equal or not.

| Operator | Description | Example |  |
| --- | --- | --- | --- |
| == | Returns true if operands are equal otherwise false. | x == y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-FUCaKJ) |
| != | Returns true if operands are not equal otherwise false. | x != y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-MXvsl7) |

Boolean Logical Operators

The Boolean logical operators perform a logical operation on bool operands.

| Operator | Description | Example |  |
| --- | --- | --- | --- |
| ! | Reverses the bool result of bool expression. Returns false if result is true and returns true if result is false. | !false | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-9l4Mbq) |
| && | Computes the logical AND of its bool operands. Returns true both operands are true, otherwise returns false. | x && y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-s7QIlQ) |
| || | Computes the logical OR of its bool operands. Returns true when any one operand is true. | x || y; | [Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-HkExSJ) |

Operator Evaluation & Precedence

Evaluation of the operands in an expression starts from left to right. If multiple operators are used in an expression, then the operators with higher priority are evaluated before the operators with lower priority.

The following table lists operators starting with the higher precedence operators to lower precedence operators.

| Operators | Category |
| --- | --- |
| x.y, x?.y, x?[y], f(x), a[i], x++, x--, new, typeof, checked, unchecked, default, nameof, delegate, sizeof, stackalloc, x->y | Primary |
| +x, -x, !x, ~x, ++x, --x, ^x, (T)x, await, &x, \*x, true and false | Unary |
| x..y | Range |
| x \* y, x / y, x % y | Multiplicative |
| x + y, x � y | Additive |
| x << y, x >> y | Shift |
| x < y, x > y, x <= y, x >= y, is, as | Relational and type-testing |
| x == y, x != y | Equality |
| x & y | Boolean logical AND |
| x ^ y | Boolean logical XOR |
| x | y | Boolean logical OR |
| x && y | Conditional AND |
| x || y | Conditional OR |
| x ?? y | Null-coalescing operator |
| c ? t : f | Conditional operator |
| x = y, x += y, x -= y, x \*= y, x /= y, x %= y, x &= y, x |= y, x ^= y, x <<= y, x >>= y, x ??= y, => | Assignment and lambda declaration |

The following example demonstrates operator precedence:

Example: Operator Precedence

int a = 5 + 3 \* 3;

int b = 5 + 3 \* 3 / 2;

int c = (5 + 3) \* 3 / 2;

int d = (3 \* 3) \* (3 / 3 + 5);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-K3cPHv)