**C# Concepts**

C# Introduction

**.NET Framework** is a development platform for building an apps for windows, web, azure, etc. by using programming languages such as C#, F# and Visual Basic. It consists of a two major components such as **Common Language Runtime** (CLR), it’s an execution engine that handles running apps and **.NET Framework Class Library**, which provides a library of tested and reusable code that developers can use it in their applications.

## **History of C#**

The C# programming language has been implemented by **Anders Hejlsberj**, the employee of Microsoft and the initial release of C# programming language is on **2002** with **.NET Framework 1.0** and it’s more like Java programming.

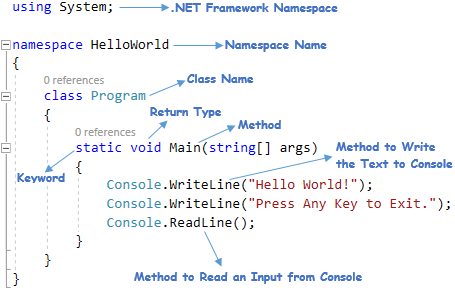
Following table lists the evaluation of c# programming language with multiple features along with .NET Framework and visual studio.

| **Version** | **Year** | **.NET Framework** | **Visual Studio** | **Features** |
| --- | --- | --- | --- | --- |
| C# 1.0 | 2002 | 1.0 / 1.1 | Visual Studio 2002 | Basic Features |
| C# 2.0 | 2005 | 2.0 | Visual Studio 2005 | * Generics * Partial types * Anonymous methods * Nullable types * Iterators * Covariance and contravariance |
| C# 3.0 | 2007 | 3.0 / 3.5 | Visual Studio 2008 | * Auto implemented properties * Anonymous types * Query expressions * Lambda expression * Expression trees * Extension methods |
| C# 4.0 | 2010 | 4.0 | Visual Studio 2010 | * Dynamic binding * Named/optional arguments * Generic covariant and contravariant * Embedded interop types |
| C# 5.0 | 2012 | 4.5 | Visual Studio 2012 / 13 | * Asynchronous members * Caller info attributes |
| C# 6.0 | 2015 | 4.6 | Visual Studio 2015 | * Static imports * Exception filters * Property initializers * Expression bodied members * Null propagator * String interpolation * nameof operator * Dictionary initializer |
| C# 7.0 | 2017 | .NET Core | Visual Studio 2017 | * Out variables * Tuples and deconstruction * Pattern matching * Local functions * Expanded expression bodied members * Ref locals and returns |

# C# Hello World Program Example

## **Explanation of C# Hello World Program**

Following diagram will illustrate the type of parameters which we used in our c# program in detailed manner.



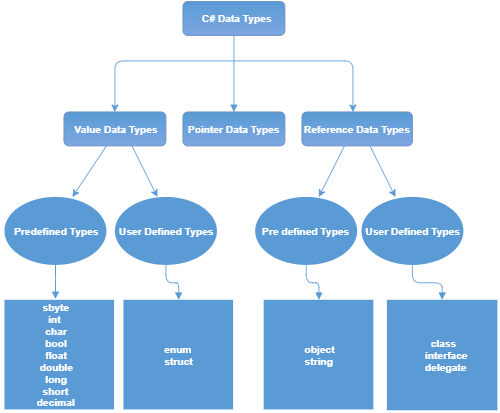
# C# Data Types with Examples

C# is a **Strongly Typed** programming language so before we perform any operation on [variables](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples), it’s mandatory to define a variable with required data type to indicate what type of data that variable can hold in our application.

## **Different Data Types in C#**

In C# programming language, we have a 3 different type of data types, those are

| **Type** | **Data Types** |
| --- | --- |
| Value Data Type | int, bool, char, double, float, etc. |
| Reference Data Type | string, class, object, interface, delegate, etc. |
| Pointer Data Type | Pointers. |



## **C# Value Data Types**

In c#, the **Value Data Types** will directly store the [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) value in memory. In c#, the value data types will accept both signed and unsigned literals.

Following table lists the value data types in c# programming language with memory size and range of values.

| **Data Type** | **.NET Type** | **Size** | **Range** |
| --- | --- | --- | --- |
| byte | Byte | 8 bits | 0 to 255 |
| sbyte | SByte | 8 bits | -128 to 127 |
| int | Int32 | 32 bits | -2,147,483,648 to 2,147,483,647 |
| uint | UInt32 | 32 bits | 0 to 4294967295 |
| short | Int16 | 16 bits | -32,768 to 32,767 |
| ushort | UInt16 | 16 bits | 0 to 65,535 |
| long | Int64 | 64 bits | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| ulong | UInt64 | 64 bits | 0 to 18,446,744,073,709,551,615 |
| float | Single | 32 bits | -3.402823e38 to 3.402823e38 |
| double | Double | 64 bits | -1.79769313486232e308 to 1.79769313486232e308 |
| bool | Boolean | 8 bits | True or False |
| decimal | Decimal | 128 bits | (+ or -)1.0 x 10e-28 to 7.9 x 10e28 |
| DateTime | DateTime | - | 0:00:00am 1/1/01 to 11:59:59pm 12/31/9999 |

## **C# Reference Data Types**

In c#, the **Reference Data Types** will contain a memory address of variable value because the reference types won’t store the variable value directly in memory.

Following table lists the reference data types in c# programming language with memory size and range of values.

| **Data Type** | **.NET Type** | **Size** | **Range** |
| --- | --- | --- | --- |
| string | String | Variable Length | 0 to 2 billion Unicode characters |
| object | Object | - | - |

## **C# Pointer Data Types**

In c#, the **Pointer Data Types** will contain a memory address of [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) value. To get the pointer details we have a two symbols ampersand (&) and asterisk (\*) in c# language.

Following is the syntax of declaring the pointer type in c# programming language.

type\* test;

Following is the example of defining the pointer type in c# programming language.

int\* a;

int\* b;

Following table lists the detail about different type pointer symbols available in c# programming language.

| **Symbol** | **Name** | **Description** |
| --- | --- | --- |
| & | Address Operator | It's useful to determine the address of a  variable. |
| \* | Indirection Operator | It's useful to access the value of an  address. |

# C# Variables with Examples

C# is a **Strongly Typed** programming language so before we perform any operation on variables, it’s mandatory to define a variable with required [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) to indicate what type of data that variable can hold in our application.

## **Rules to Declare a C# Variables**

* We can define a variable name with the combination of alphabets, numbers and underscore.
* A variable name must always starts with either alphabets or underscore but not with numbers.
* While defining the variable, no white space is allowed within the variable name.
* We should not use any reserve keywords such as int, float, char, etc. for variable name.
* In c#, once the variable is declared with a particular [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples), it cannot be re-declared with a new type and we shouldn’t assign a value that is not compatible with the declared type.

# C# Value Type and Reference Type with Examples

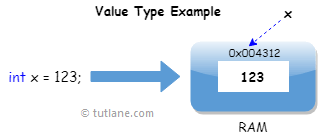
In c#, we have a two ways to allocate the space in memory, i.e. either on **stack** or **heap** memory based on the **Value Type** or **Reference Type** parameters.

## **C# Value Types**

In c#, a [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) is a **Value Type** if it hold the value of [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) directly on its own memory space and Value Types will use **Stack** memory to store the variables values.

For example, if we define and assign a value to the variable like int x = 123; then the system will use the same memory space of variable ‘**x**’ to store the value ‘**123**’.

Following is the pictorial representation of value types in c# programming language.



Following are the different data types which will fall under **Value Type** in c# programming language.

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

## **C# Pass Value Type by Value**

In c#, if we pass a value type variable from one method to another method, the system will create a separate copy for the variable in another method. In case, if we make a changes to the variable in one method won’t affect the variable in other method.

Following is the example of passing the value type by value in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Square(int a, int b)

        {

            a = a \* a;

            b = b \* b;

            Console.WriteLine(a + " " + b);

        }

        static void Main(string[] args)

        {

            int num1 = 5;

            int num2 = 10;

            Console.WriteLine(num1 + " " + num2);

            Square(num1, num2);

            Console.WriteLine(num1 + " " + num2);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

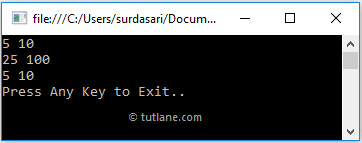
        }

    }

}

If you observe above example, we defined a two variables (**num1**, **num2**) in **Main()** method and we are making a changes to those variable by passing it to **Square()** method but those changes won’t affect the variables in **Main()** method.

When we execute above program, we will get the result like as shown below.



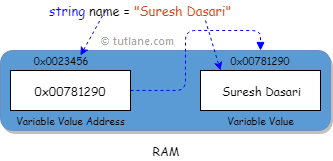
If you observe above result, the changes made to the variables in **Square()** method didn’t affected the variables in **Main()** method.

## **C# Reference types**

In c#, **Reference Types** will contain a pointer which points to other memory location that holds the data. The **Reference Types** won’t store the variable value directly in its memory instead, it will store the memory address of the variable value to indicate where the value is being stored.

For example, if we define and assign a value to the variable like string name = "Suresh Dasari"; then the system will store the variable value “**Suresh Dasari**” in one location and the variable "**name**" in another location along with the memory address of the variable value.

Following is the pictorial representation of reference type in c# programming language.



Following are the different data types which will fall under **Reference Type** in c# programming language.

* [String](https://www.tutlane.com/tutorial/csharp/csharp-string-with-examples)
* [Class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples)
* [Delegates](https://www.tutlane.com/tutorial/csharp/csharp-delegates)
* All [Arays](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples), Even if their elements are value types

## **Pass Reference Type by Value**

In c#, if we pass a reference type variable from one method to another method, the system won’t create a separate copy for that variable instead it passes the address of the variable. So if we make any changes to the variable in one method that also reflect in other method.

Following is the example of passing the reference type by value in c# programming language.

using System;

namespace CsharpExamples

{

    class Person

    {

        public int age;

    }

    class Program

    {

        static void Square(Person a, Person b)

        {

            a.age = a.age \* a.age;

            b.age = b.age \* b.age;

            Console.WriteLine(a.age + " " + b.age);

        }

        static void Main(string[] args)

        {

            Person p1 = new Person();

            Person p2 = new Person();

            p1.age = 5;

            p2.age = 10;

            Console.WriteLine(p1.age + " " + p2.age);

            Square(p1, p2);

            Console.WriteLine(p1.age + " " + p2.age);

            Console.WriteLine("Press Any Key to Exit..");

            Console.ReadLine();

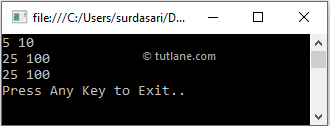
        }

    }

}

If you observe above example, we created a new class called “**Person**” and created an instance of new class (**Person**) and assigned a values to the variables in **Main()** method and we are making a changes to those variable by passing it to **Square()** method and those changes will be reflected in **Main()** method.

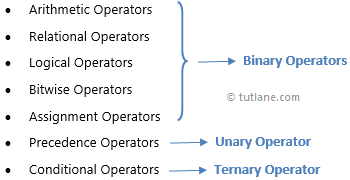
When we execute above program, we will get the result like as shown below.



If you observe above result, the changes whatever we made to the variables in **Square()** method, that also reflected for the variables in **Main()** method.

# C# Operators (Arithmetic, Relational, Logical, Assignment, Precedence)

In c#, we have a different type of operators available, those are



Suppose, if operator that takes a one operand to perform the operation such as precedence operator (++), then those will call as a **Unary Operators**. In case, if operator that takes a two operands to perform the operation such as [arithmetic operator](https://www.tutlane.com/tutorial/csharp/csharp-arithmetic-operators-with-examples) (-, +, \*, /), then those will call as a **Binary Operators** and we have a [Ternary Operator](https://www.tutlane.com/tutorial/csharp/csharp-ternary-operator-with-examples) that takes three operands to perform the operation, such as [conditional operator](https://www.tutlane.com/tutorial/csharp/csharp-ternary-operator-with-examples) (**?:**).

# C# Arithmetic Operators with Examples

In c#, **Arithmetic Operators** are used to perform basic arithmetic calculations like addition, subtraction, division, etc. based on our requirements.

For example, we have an integer variables **x = 20**, **y = 10** and if we apply an arithmetic operator **+** (**x + y**) to perform an addition operator, then we will get the result as **30**like as shown below.

int result;

int x = 20, y = 10;

result = (x + y);

Following table lists the different type of operators available in c# arithmetic operators.

| **Operator** | **Name** | **Description** | **Example (a = 6, b = 3)** |
| --- | --- | --- | --- |
| + | Addition | It adds two operands. | a + b = 9 |
| - | Subtraction | It subtract two operands. | a - b = 3 |
| \* | Multiplication | It multiplies two operands. | a \* b = 18 |
| / | Division | It divides numerator by de-numerator. | a / b = 2 |
| % | Modulo | It returns a remainder as result. | a % b = 0 |

| **Operator** | **Name** | **Description** | **Example (a = 6, b = 3)** |
| --- | --- | --- | --- |
| == | Equal to | It compare two operands and return true if both are same. | a == b (false) |
| > | Greater than | It compare whether left operand greater  than right operand or not and return true if it satisfied. | a > b (true) |
| < | Less than | It compare whether left operand less than  right operand or not and return true if it satisfied. | a < b (false) |
| >= | Greater than or Equal to | It compare whether left operand greater  than or equal to right operand or not and return true if it satisfied. | a >= b (true) |
| <= | Less than or Equal to | It compare whether left operand less than  or equal to right operand or not and return true if it satisfied. | a <= b (false) |
| != | Not Equal to | It checks whether two operand values equal  or not and return true if values are not equal. | a != b (true) |

# C# Relational Operators with Examples

In c#, **Relational Operators** are used to check the relation between two operands like we can determine whether two operand values equal or not, etc. based on our requirements.

Generally, in c# the relational operators will return **true** only when the defined operands relationship become **true**, otherwise it will return **false**.

For example, we have an integer variables **a = 10**, **b = 20** and if we apply a relational operator **>=** (**a >= b**), we will get the result **false** because the variable “**a**” contains a value which is less than variable **b**.

Following table lists the different type of operators available in c# relational operators.

# C# Logical Operators with Examples

In c#, **Logical Operators** are used to perform the logical operation between two operands like AND, OR and NOT based on our requirements. The Logical Operators will always work with Boolean expressions (**true** or **false**) and return Boolean values.

The operands in logical operators must always contain only Boolean values otherwise Logical Operators will throw an error.

Following table lists the different type of operators available in c# relational operators.

| **Operator** | **Name** | **Description** | **Example (a = true, b = false)** |
| --- | --- | --- | --- |
| && | Logical AND | It return true if both operands are non zero. | a && b (false) |
| || | Logical OR | It returns true if any one operand become a non zero. | a || b (true) |
| ! | Logical NOT | It return the reverse of logical state that means if both  operands are non zero then it will return false. | !(a && b) (true) |

# C# Bitwise Operators with Examples

In c#, **Bitwise Operators** will work on bits and these are useful to perform a bit by bit operations such as Bitwise AND (&), Bitwise OR (|), Bitwise Exclusive OR (^), etc. on operands and we can perform bit level operations on Boolean and integer data.

For example, we have an integer variables **a = 10**, **b = 20** and the binary format of these variables will be like as shown below.

a = 10 (00001010)

b = 20 (00010100)

When we apply **Bitwise OR** (**|**) operator on these parameters we will get the result like as shown below.

00001010

00010100

-----------

00011110 = 30 (Decimal)

Following table lists the different type of operators available in c# bitwise operators.

| **Operator** | **Name** | **Description** | **Example (a = 0, b = 1)** |
| --- | --- | --- | --- |
| & | Bitwise AND | It compares each bit of first operand with the corresponding bit of its sencond operand. If both bits are 1, then the result bit will be 1 otherwise the result bit will be 0. | a & b (0) |
| | | Bitwise OR | It compares each bit of first operand with the corresponding bit of its sencond operand. If either of bit is 1, then the result bit will be 1 otherwise the result bit will be 0. | a | b (1) |
| ^ | Bitwise Exclusive OR (XOR) | It compares each bit of first operand with the corresponding bit of its sencond operand. If one bit is 0 and other bit is 1, then the result bit will be 1 otherwise the result bit will be 0. | a ^ b (1) |
| ~ | Bitwise Complement | It operates on only one operand and it will invert each bit of operand. It will change bit 1 to 0 and vice versa. | ~(a) (1) |
| << | Bitwise Left Shift) | It shifts the number to the left based on the specified number of bits. The zeroes will be added to the least significant bits. | b << 2 (100) |
| >> | Bitwise Right Shift | It shifts the number to the right based on the specified number of bits. The zeroes will be added to the least significant bits. | b >> 2 (001) |

# C# Assignment Operators with Examples

In c#, **Assignment Operators** are used to assign a new value to the operand and these operators will work with only one operand.

For example, we can declare and assign a value to the variable using assignment operator (**=**) like as shown below.

int a;

a = 10;

If you observe above sample we defined a variable called “**a**” and assigned a new value using assignment operator (**=**) based on our requirements.

Following table lists the different type of operators available in c# assignment operators.

| **Operator** | **Name** | **Description** | **Example** |
| --- | --- | --- | --- |
| = | Equal to | It is used to assign the values to variables. | int a; a = 10 |
| += | Addition Assignment | It perform an addition of left and right operands and assign a result to the left operand. | a += 10 is equals to a = a + 10 |
| -= | Subtraction Assignment | It perform a subtraction of left and right operands and assign a result to the left operand. | a -= 10 is equals to a = a - 10 |
| \*= | Multiplication Assignment | It perform a multiplication of left and right operands and assign a result to the left operand. | a \*= 10 is equals to a = a \* 10 |
| /= | Division Assignment | It perform a divison of left and right operands and assign a result to the left operand. | a /= 10 is equals to a = a / 10 |
| %= | Modulo Assignment | It perform a modulo operation on two operands and assign a result to the left operand. | a %= 10 is equals to a = a % 10 |
| &= | Bitwise AND Assignment | It perform a Bitwise AND operation on two operands and assign a result to the left operand. | a &= 10 is equals to a = a & 10 |
| |= | Bitwise OR Assignment | It perform a Bitwise OR operation on two operands and assign a result to the left operand. | a |= 10 is equals to a = a | 10 |
| ^= | Bitwise Exclusive OR Assignment | It perform a Bitwise XOR operation on two operands and assign a result to the left operand. | a ^= 10 is equals to a = a ^ 10 |
| >>= | Right Shift Assignment | It moves the left operand bit values to right based on the number of positions specified by second operand. | a >>= 2 is equals to a = a >> 2 |
| <<= | Left Shift Assignment | It moves the left operand bit values to left based on the number of positions specified by second operand. | a <<= 2 is equals to a = a << 2 |

# C# Operators Precedence with Examples

In c#, **Operators Precedence** is used to define a multiple operators in single expression and the evaluation of expression can be happened based on the priority of operators.

For example, the multiplication operator (**\***) is having a higher precedence than the addition operator (**+**). So if we use both multiplication (**\***) and addition (**+**) operators in single expression, first it will evaluate the multiplication part and then the addition part in expression.

Following is the simple example of defining an expression with operator precedence in c#.

int i = 3 + 4 \* 5;

When we execute above statement, first the multiplication part (**4 \* 5**) will be evaluated. After that the addition part (**3 + 12**) will be executed and i value will become a **23**.

As said earlier, multiplication (**\***) operator is having a higher precedence than addition (**+**) operator so first multiplication part will be executed.

Following table lists the different type of operators available in c# relational operators.

| **Category** | **Operator(s)** |
| --- | --- |
| Postfix / Prefix | ++, -- |
| Unary | +, -, !, ~ |
| Multiplicative | \*, /, % |
| Additive | +, - |
| Shift | <<, >> |
| Relational | <, <=, >, >= |
| Equality | ==, != |
| Bitwise | &, |, ^ |
| Logical | &&, || |
| Conditional | ?: |
| Assignment | =, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>= |

## **C# Operator Precedence Example**

Following is the example of implementing an operator precedence in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x = 20, y = 5, z = 4;

            int result = x / y + z;

            Console.WriteLine("Result1: "+result);

            bool result2 = z <= y + x;

            Console.WriteLine("Result2: "+result2);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

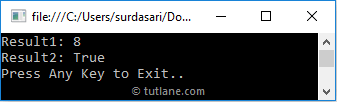
    }

}

If you observe above example, we are implemented an operator precedence with multiple operators and the expressions evaluation will be done based on the priority of operators in c# programming language.

## **Output of C# Operators Precedence Example**

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



# C# If Statement with Examples

## **Syntax of C# if Statement**

Following is the syntax of defining if statement in c# programming language.

if (bool\_expression) {

// Statements to Execute if condition is true

}

# C# If Else Statement with Examples

## **Syntax of C# if-else Statement**

Following is the syntax of defining **if else** statement in c# programming language.

if (boolean\_expression)

{

// Statements to Execute if boolean expression is True

}

else {

// Statements to Execute if boolean expression is False

}

# C# If-Else-If Statement with Examples

## **Syntax of C# if-else-if Statement**

Following is the syntax of defining **if else if** statement in c# programming language.

if (condition\_1)

{

// Statements to Execute if condition\_1 is True

}

else if (condition\_2)

{

// Statements to Execute if condition\_2 is True

}

else if (condition\_3)

{

// Statements to Execute if condition\_3 is True

}

....

....

else{

// Statements to Execute if all conditions are False

}

# C# Nested If Else Statements with Examples

## **Syntax of C# Nested If-Else Statement**

Following is the syntax of defining nested [if…else statement](https://www.tutlane.com/tutorial/csharp/csharp-if-else-statement-with-examples) in c# programming language.

if (condition)

{

if (nested\_condition\_1)

{

// Statements to Execute

}

else

{

// Statements to Execute

}

}

else

{

if (nested\_condition\_2)

{

// Statements to Execute

}

else

{

// Statements to Execute

}

}

# C# Ternary Operator (?:) with Examples

## **Syntax of C# Ternary Operator**

In c#, the Ternary Operator will always work with **3** operands. Following is the syntax of defining a Ternary Operator in c# programming language.

condition\_expression ? first\_expression : second\_expression;

In c#, the Ternary Operator (?:) will work like as follow.

* In Ternary Operator, the **condition expression** must be evaluated to either **true** or **false**. If **condition** is **true**, the **first\_expression** result returned by the ternary operator.
* In case, if **condition** is **false**, then the **second\_expression** result returned by the operator.

The Ternary Operator (?:) is a substitute of [if…else statement](https://www.tutlane.com/tutorial/csharp/csharp-if-else-statement-with-examples) in c# programming language.

## **C# Nested Ternary Operator**

int x = 20, y = 20;

// If...else If Statement

string result;

if (x > y)

{

result = "x value greater than y";

}

else if (x < y)

{

result = "x value less than y";

}

else {

result = "x value equals to y";

}

//Nested Ternary Operator (?:)

result = (x > y) ? "x value greater than y" : (x < y) ? "x value less than y" : "x value equals to y";

If you observe above code, we are able to replace multiple lines of [if…else if](https://www.tutlane.com/tutorial/csharp/csharp-if-else-if-statement-with-examples) code with single line of nested ternary operator based on our requirements.

In c#, the conditional operator is a right associative so the expression **a ? b : c ? d : e;** evaluated as **a ? b : (c ? d : e)**, not as **(a ? b : c) ? d : e**.

# C# Switch Case Statement with Examples

## **Syntax of C# Switch Statement**

Generally, in c# switch statement is a collection of multiple case statements and it will execute only one single case statement based on the matching value of expression.

Following is the syntax of defining the switch statement in c# programming language.

switch(variable/expresison){

case value1:

// Statements to Execute

break;

case value2:

//Statements to Execute

break;

....

....

default:

// Statements to Execute if No Case Matches

break;

}

## **C# Nested Switch Case Statements**

In c#, using one **switch** statement within another **switch** statement is called a **nested switch case** statements.

Following is the example of using nested switch statements in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x = 10, y = 5;

            switch (x)

            {

                case 10:

                    Console.WriteLine("X Value: 10");

                    switch(y){

                        case 5:

                            Console.WriteLine("Nested Switch Value: 5");

                            switch (y - 2) {

                                case 3:

                                    Console.WriteLine("Another Nested Switch Value: 3");

                                    break;

                            }

                            break;

                    }

                    break;

                case 15:

                    Console.WriteLine("X Value: 15");

                    break;

                case 20:

                    Console.WriteLine("X Value: 20");

                    break;

                default:

                    Console.WriteLine("Not Known");

                    break;

            }

            Console.WriteLine("Press Enter Key to Exit..");

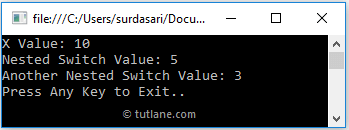
            Console.ReadLine();

        }

    }

}

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



## **C# Switch Case Statement with Enum**

In c#, we can use **enum** values with Switch case statements to perform required operations.

Following is the example of using **enum** values in c# **switch case** statement.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            location loc = location.hyderabad;

            switch (loc)

            {

                case location.chennai:

                    Console.WriteLine("Location: Chennai");

                    break;

                case location.guntur:

                    Console.WriteLine("Location: Guntur");

                    break;

                case location.hyderabad:

                    Console.WriteLine("Location: Hyderabad");

                    break;

                default:

                    Console.WriteLine("Not Known");

                    break;

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public enum location

        {

            hyderabad,

            chennai,

            guntur

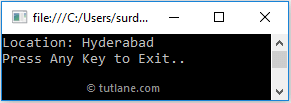
        }

    }

}

If you observe above example, we defined a enum values and using those values in switch case statements to perform required operations based on our requirements.

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



# C# For loop with Examples

## **Syntax of C# For Loop**

Following is the syntax of defining for loop in c# programming language.

for (initialization; condition; iterator(inc / dec))

{

// Statements to Execute

}

## **C# For Loop with Multiple Variables**

In c# for loop, we can declare and initialize multiple variables and iterator expressions by separating with **comma** (**,**) operator.

Following is the example of using multiple variables and iterator expressions in c# **for** loop.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1, j = 0; i <= 4; i++, j++)

            {

                Console.WriteLine("i: {0}, j: {1}", i, j);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

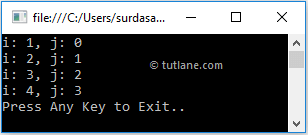
        }

    }

}

If you observe above example, we defined a two variables (**i**, **j**) and two iterator expressions (**i++**, **j++**) by separating them with **comma** (**,**) operator.

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



## **C# For Loop with Break Statement**

In c#, by using **break** keyword we can stop the execution of **for** loop statement based on our requirements.

Following is the example of stop the execution of **for** loop using **break** statement.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1; i <= 4; i++)

            {

                if (i == 3)

                    break;

                Console.WriteLine("i value: {0}", i);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

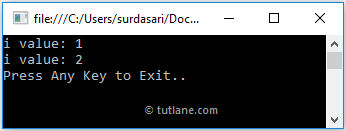
        }

    }

}

If you observe above code, we used a **break** statement to **exit** for loop whenever the variable i value equals to **3**.

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



## If you observe above result, whenever the variable **i** value equals to **3**, then automatically the **for** loop execution has stopped.

## **C# For Loop without Initialization & Iterators**

Generally, the initializer, condition and iterator parameters are optional to create **for** loop in c# programming language.

Following is the example of creating a **for** loop in c# programming language without **initializer** and **iterator**.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int i = 1;

            for ( ; i <= 4; )

            {

                i++;

                Console.WriteLine("i value: {0}", i);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

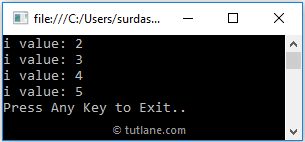
        }

    }

}

If you observe above example, we defined a for loop without any initializers and iterators.

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



## **C# Infinite For Loop**

In case, if the **condition** parameter in **for** loop always returns true, then the **for** loop will be an **infinite** and runs forever. Even if we miss the condition parameter in for loop automatically that loop will become an infinite loop.

Following are the different ways to make for loop as an infinite loop in c# programming language.

for (initializer; ; iterator) {

// Statements to Execute

}

or

for ( ; ; )

{

// Statements to Execute

}

Following is the example of making a for loop as an infinite in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1; i > 0; i++)

            {

                i++;

                Console.WriteLine("i value: {0}", i);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

    }

}

If you observe above code, the condition (**i > 0**) whatever we defined in for loop will always returns true so it will return infinite result.

## **C# Nested For Loop**

In c#, we can create one for loop within another for loop based on our requirements. Following is the example of creating a nested for loop in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1; i <= 4; i++)

            {

                for (int j = i; j < 3; j++)

                {

                    Console.WriteLine("i value: {0}, j value: {1}", i, j);

                }

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

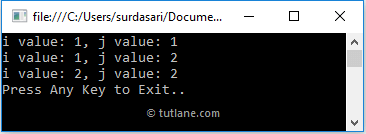
        }

    }

}

If you observe above example, we created a for loop within another loop and printing the values based on our requirements.

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



# C# While Loop with Examples

## **Syntax of C# While Loop**

Generally, **while** keyword is used to create a while loop in c# applications. Following is the syntax of defining a while loop in c# programming language to execute the block of statements till the defined condition evaluates as **false**.

while (boolean\_expression) {

// Statements to Execute

}

# C# Do While Loop with Examples

In c#, **Do-While** loop is used to execute a block of statements until the specified expression return as a true.

Generally, in c# the **do-while** loop is same as [while loop](https://www.tutlane.com/tutorial/csharp/csharp-while-loop-with-examples) but only the difference is [while loop](https://www.tutlane.com/tutorial/csharp/csharp-while-loop-with-examples) will execute the statements only when the defined condition returns **true** but **do-while** loop will execute the statements at least once, because first it will execute the block of statements and then it will checks the condition.

## **Syntax of C# Do-While Loop**

Generally, **do** and [**while**](https://www.tutlane.com/tutorial/csharp/csharp-while-loop-with-examples) keywords are used to create a **do...while** loop in C#. Following is the syntax of defining a **do-while** loop in c# programming language to execute the block of statements till the defined condition evaluates as **false**.

do

{

// Statements to Execute

}while (boolean\_expression);

# C# Foreach Loop with Examples

## **Syntax of C# Foreach Loop**

Following is the syntax of defining the **Foreach** loop in c# programming language.

foreach (Type var\_name in Collection\_Object) {

// Statements to Execute

}

Here, **Type** is a built in [data-type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) or custom class type and **var\_name** is a [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) name to access an elements from collection object (**Collection\_Object**) to use it in body of the foreach loop.

## **C# Foreach Loop with Array Example**

Following is the example of using **foreach** loop in c# programming language to iterate or loop through [array](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) elements.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            string[] names = new string[3] { "Suresh Dasari", "Rohini Alavala", "Trishika Dasari" };

            foreach (string name in names)

            {

                Console.WriteLine(name);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

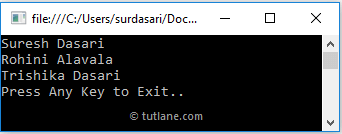
    }

}

If you observe the above example, we created a string array object “**names**” and looping through an each element of [array](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) object using **foreach** loop and assigning an array elements to string variable “**name**”.

To know more about arrays in c# programming language, check this [c# arrays with examples](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples).

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



# C# Break Statement with Examples

In c#, **Break** statement is used to break or terminate the execution of loops (for, while, do-while, etc.) or switch statement and the control is passed immediately to the next statements that follows a terminated loops or statements.

# C# Continue Statement with Examples

In c#, **Continue** statement is used to pass a control to the next iteration of loops such as [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples), [while](https://www.tutlane.com/tutorial/csharp/csharp-while-loop-with-examples), [do-while](https://www.tutlane.com/tutorial/csharp/csharp-do-while-loop-with-examples) or [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples) from the specified position by skipping the remaining code.

In previous section, we learned [break statement in c#](https://www.tutlane.com/tutorial/csharp/csharp-break-statement-with-examples). The main difference between [break](https://www.tutlane.com/tutorial/csharp/csharp-break-statement-with-examples) statement and **continue** statement is, the [break](https://www.tutlane.com/tutorial/csharp/csharp-break-statement-with-examples) statement will completely terminate the loop or statement execution but the **continue** statement will pass a control to the next iteration of loop.

## **C# For Loop with Continue Statement**

In c#, by using **continue** keyword we can skip the execution of further code and send back control to next iteration of [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) loop statement based on our requirements.

Following is the example of using **continue** statement with [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) loop in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1; i <= 4; i++)

            {

                if (i == 3)

                    continue;

                Console.WriteLine("i value: {0}", i);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

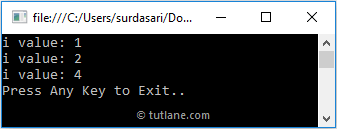
        }

    }

}

If you observe above code, we used a **continue** statement to pass control back to the next iteration of [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) loop whenever the variable **i** value equals to **3**.

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



If you observe above result, whenever the variable **i** value equals to **3**, it skips the further execution of statements and passes the control back to the next iteration of [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) loop.

# C# Goto Statement with Examples

In c#, **Goto** statement is used to transfer a program control to the defined labeled statement and it is useful to get out of the loop or exit from a deeply nested loops based on our requirements.

Generally, in c# the defined labeled statement must always exists in the scope of **goto** statement and we can define multiple **goto** statements in our application to transfer the program control to specified labeled statement.

For example, we can use **goto** statement in [switch](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) statement to transfer a control from one [switch-case](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) label to another or to a default label based on our requirements.

## **Syntax of C# Goto Statement**

Following is the syntax of defining a **goto** statement in c# programming language.

goto labeled\_statement;

If you observe above syntax we defined a **goto** statement by using **goto** keyword and with **labeled\_statement**. Here the **labeled\_statement** is used to transfer the program control to specified **labeled\_statement** position.

## **C# Goto Statement with For Loop Example**

Following is the example of using **goto** statement in [for loop](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) to exit the loop based on our requirements.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            for (int i = 1; i < 10; i++)

            {

                if (i == 5)

                {

                    goto endloop;

                }

                Console.WriteLine("i value: {0}", i);

            }

            endloop: Console.WriteLine("The end");

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

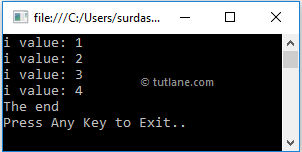
        }

    }

}

If you observe above example, we used a **goto** in [for loop](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) with labeled statement “**endloop**” to exit for loop whenever the variable (**i**) value equals to **5**.

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



If you observe above result, whenever the variable (**i**) value equals to **5**, the **goto** statement transferred the program control from [for loop](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) to specified label statement (**endloop**) position.

## **C# Goto Statement with Switch Statement**

In c#, we can use **goto** statement exit from defined loops or transfer a control to specific [switch-case](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) label or the default label in [switch](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) statement based on our requirements.

Now we will see how to use **goto** statement in [switch-case](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) statement with example. Following is the example of using **goto** with [switch-case](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) statement to transfer a control from one [switch-case](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) label to another based on our requirements.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int i = 3, j = 0;

            switch (i)

            {

                case 1:

                    j += 20;

                    Console.WriteLine("j value is {0}",j);

                    break;

                case 2:

                    j += 5;

                    goto case 1;

                case 3:

                    j += 30;

                    goto case 1;

                default:

                    Console.WriteLine("Not Known");

                    break;

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

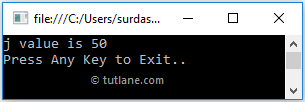
        }

    }

}

If you observe above example, we used a **goto** statement in multiple [switch](https://www.tutlane.com/tutorial/csharp/csharp-switch-case-statement-with-examples) cases and trying to transfer program control from **case 2** / **case 3** to **case 1**.

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



# C# Return Statement with Examples

In c#, **Return** statement is used to terminate the execution of method in which it appears and returns the control back to the calling method.

Generally, in c# the **return** statement is useful whenever we want to get a some value from the other methods and we can omit the usage of **return** statement in our methods by using **void** as a return type.

# C# Methods / Functions with Examples

In c#, **Method** is a separate code block and that contain a series of statements to perform a particular operations and methods must be declared either in **class** or **struct** by specifying the required parameters.

Generally, in c# Methods are useful to improve the code reusability by reducing the code duplication. Suppose if we have a same functionality to perform in multiple places, then we can create a one method with required functionality and use it wherever it is required in application.

## **Syntax of C# Methods**

As discussed, in c# **Methods** must be declared either in a **class** or **struct** by specifying the required access level, return type, name of the method and any method parameters like as shown below.

class class\_name

{

    ...

    ...

    <Access\_Specifier> <Return\_Type> Method\_Name(<Parameters>)

    {

        // Statements to Execute

    }

    ...

    ...

}

If you observe the above syntax, we defined a method in **class** with various parameters, those are

**Access\_Specifier** - It is used to define an access level either **public** or **private**, etc. to allow other classes to access the method. If we didn’t mention any access modifier, then by default it is **private**.

**Return\_Type** - It is used to specify the type of value the method can return. In case, if method is not returning any value, then we need to mention **void** as return type.

**Method\_Name** - It must be a unique name to identify the method in a class.

**Parameters** - The method parameters are used to send or receive a data from method and these method parameters are enclosed within parentheses and are separated by commas. In case, if no parameters are required for a method then, we need to define a method with empty parentheses.

 In c#, both **methods** and **functions** are same, there is no difference and these are a just different terms to do the same thing in c#.

## **C# Passing Parameters to Methods**

In c#, we have a different ways to pass a parameters to the method, those are

| **Parameters** | **Description** |
| --- | --- |
| [Value Parameters](https://www.tutlane.com/tutorial/csharp/csharp-pass-by-value-with-examples) | These are called as “**input parameters**” and these parameters will pass a copy of original value instead of  original parameters. So the changes made to the parameters in called method will not have an effect on the  original values when control returns to the caller. |
| [Reference Parameters](https://www.tutlane.com/tutorial/csharp/csharp-pass-by-reference-ref-with-examples) | These are called as “**input/output parameters**” and these will pass a memory reference of original  parameters. So the changes made to the parameters in called method will have an effect on the  original values when control returns to the caller. |
| [Output Parameters](https://www.tutlane.com/tutorial/csharp/csharp-out-parameter-with-examples) | These are called as “**output parameters**” and these are more like reference type parameters but only  difference is we don’t need to initialize it before passing. |

# C# Pass By Value with Examples

In c#, Passing a [Value-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) parameter to a method by value means passing a copy of the variable to the method. So the changes made to the parameter inside of called method will not have an effect on the original data stored in the argument variable.

As discussed earlier, [Value-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) variables will contain the value directly on it memory and [Reference-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) variables will contain a reference of its data.

## **C# Passing Parameters By Value Example**

Following is the example of passing a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) parameter to a method by value in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x = 10;

            Console.WriteLine("Variable Value Before Calling the Method: {0}", x);

            Multiplication(x);

            Console.WriteLine("Variable Value After Calling the Method: {0}", x);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public static void Multiplication(int a)

        {

            a \*= a;

            Console.WriteLine("Variable Value Inside the Method: {0}", a);

        }

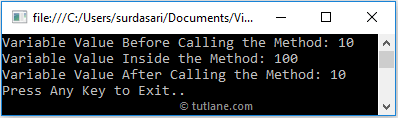
    }

}

If you observe above example, the variable **x** is a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) and it passed to the **Multiplication** method. The content of variable **x**, copied to the parameter **a** and made required modifications in **Multiplication** method but the changes that made inside of the method have no effect on the original value of the variable.

## **Output of C# Passing Parameters By Value Example**

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



If you observe above result, the variable value not changed even after we made the modifications in our method.

# C# Pass By Reference (Ref) with Examples

In c#, passing a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) parameter to a method by reference means passing a reference of the variable to the method. So the changes made to the parameter inside of called method will have an effect on the original data stored in the argument variable.

By using ref keyword, we can pass a parameters [reference-type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) and it’s mandatory to initialize the variable value before we pass it as an argument to the method in c# programming language.

## **Declaration of C# Pass By Reference**

Following is simple example of passing parameters by reference in c# programming language.

int x = 10; // Variable need to be initialized

Multiplication(ref x);

## **C# Passing Parameters By Reference Example**

Following is the example of passing a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) parameter to a method by reference in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x = 10;

            Console.WriteLine("Variable Value Before Calling the Method: {0}", x);

            Multiplication(ref x);

            Console.WriteLine("Variable Value After Calling the Method: {0}", x);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public static void Multiplication(ref int a)

        {

            a \*= a;

            Console.WriteLine("Variable Value Inside the Method: {0}", a);

        }

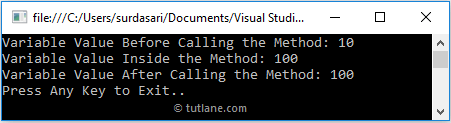
    }

}

If you observe above example, we are passing the reference of variable **x** to the variable **a** in **Multiplication** method by using **ref** keyword. In this case, the variable **a** contains the reference of variable **x** so the changes that made to the variable **a** will affect the value of variable **x**.

## **Output of C# Passing Parameters By Reference Example**

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



# C# Out Parameter with Examples

In c#, **out** keyword is used to pass an arguments to the method as a [reference type](https://www.tutlane.com/tutorial/csharp/csharp-pass-by-reference-ref-with-examples). The **out** keyword is same like **ref** keyword, only difference is **out** doesn’t require a variable to be initialized before we pass it as an argument to the method but the [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) must be initialized in called method before it return a value to the calling method.

The out parameter in c# is also useful to return more than one value from the methods in c# programming language.

## **Declaration of C# Out Parameter**

Following is simple example of using **out** parameters in c# programming language.

int x; // No need to initialize variable

Multiplication(out x);

If you observe above declaration, we just declared a variable **x** and pass it to the method by using **out** parameter without assigning any value but as discussed, the variable must be initialized in called method before it return a value to the calling method.

To use **out** parameter in c# application, both the method definition and the calling method must explicitly use the **out** keyword.

## **C# Out Parameter Example**

Following is the example of passing an **out** parameter to the method in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x;

            Multiplication(out x);

            Console.WriteLine("Variable Value: {0}", x);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public static void Multiplication(out int a)

        {

            a = 10;

            a \*= a;

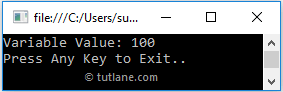
        }

    }

}

If you observe above example, we declared a variable **x** and passing it to a **Multiplication** method by using **out** keyword without initializing the value. However, the called method (**Multiplication**) is initializing the value before it returning the value to the calling method.

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



If you observe above result, the changes whatever we did for variable in **Multiplication** method has reflected in calling method also.

## **C# Multiple Out Parameters Example**

Following is the example of using multiple **out** parameters in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x, y;

            Multiplication(out x, out y);

            Console.WriteLine("x Value: {0}", x);

            Console.WriteLine("y Value: {0}", y);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public static void Multiplication(out int a, out int b)

        {

            a = 10;

            b = 5;

            a \*= a;

            b \*= b;

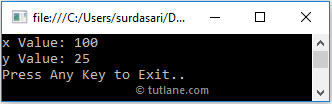
        }

    }

}

If you observe above example, we defined a two variables (x, y) and passing it to **Multiplication** method using **out** parameters.

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



If you observe above result, the changes whatever we did for variables in **Multiplication** method has reflected in calling method also.

# C# Params Keyword with Examples

In c#, **params** keyword is used to specify a method parameter that takes a variable number of arguments. The params keyword is useful when we are not sure about number of arguments to send as a parameter.

In c#, during method declaration only one params keyword is allowed and no additional parameters are permitted after the params keyword in a method declaration.

We can send an arguments of specified type as a comma-separated list or an array to the declared parameter. In case, if we are not sending any arguments to the defined parameter, then the length of params list will become a zero.

## **C# Params Keyword Example**

Following is the example of using params keyword in c# programming language to specify method parameter accept multiple number of arguments.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };

            ParamsMethod(arr);

        }

        public static void ParamsMethod(params int[] arr)

        {

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

            {

                Console.Write(arr[i] + (i < arr.Length - 1 ? ", " : ""));

            }

            Console.WriteLine();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

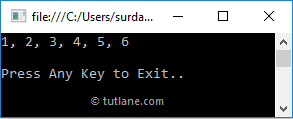
        }

    }

}

If you observe above example, we are sending a comma separated list of multiple arguments of the specified type (integer) to the declared parameter in **ParamsMethod**function.

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



If you observe above result, we are able to send multiple arguments of same data type to params keyword parameter in method declaration.

In previous example, we are sending only **integer** type of arguments to the method parameter. In case, if you want to send a list of multiple type of arguments, then we need to use object type parameter in method declaration.

## **C# Params Keyword with Object Type**

Following is the example of using object type parameter in method declaration to accept list of multiple type of arguments.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            object[] arr =  { 1, 2, "suresh", "rohini", "trishika", 10.26 };

            ParamsMethod(arr);

        }

        public static void ParamsMethod(params object[] arr)

        {

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

            {

                Console.Write(arr[i] + (i < arr.Length - 1 ? ", " : ""));

            }

            Console.WriteLine();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

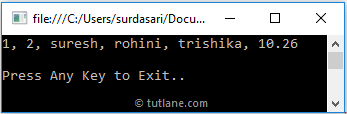
        }

    }

}

If you observe above example, we used object type parameter in method declaration and accepting different data type of arguments as a list.

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



If you observe above result, we are able to send different data type of arguments to the params keyword parameter in method declaration.

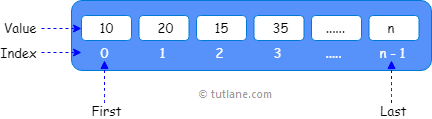
# C# Arrays with Examples

In c#, **Arrays** are useful to store a multiple elements of same data type at contiguous memory locations and an arrays will allow us to store fixed number of elements sequentially based on the predefined number of items.

In previous chapter, we learned about [variables in c#](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples), which will help us to hold a single value like **int x = 10;**. In case if we want to hold more than one value of same [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples), then an arrays came into the picture in c# to solve this problem.

An array can start storing the values from index **0**. Suppose if we have an array with n elements, then it will start storing the elements from index **0** to **n-1**.

Following is the pictorial representation of storing the multiple values of same type in c# array data structure.



## **C# Arrays Initialization**

// Declaring and Initializing an array with size of 5

int[] array = new int[5];

//Defining and assigning an elements at the same time

int[] array2 = new int[5]{1,2,3,4,5};

//Initialize with 5 elements will indicates the size of an array

int[] array3 = new int[] { 1, 2, 3, 4, 5 };

// Another way to initialize an array without size

int[] array4 = { 1, 2, 3, 4, 5 };

// Declare an array without initialization

int[] array5;

array5 = new int[]{ 1, 2, 3, 4, 5 };

If you observe above examples, in first statement we declared and initialized an integer array with the size of **5** to allow an array to store **5** integer values and the array can contain an elements from **array[0]** to **array[4]**.

Generally, in c# initializing an array without **size** or assigning a values to an array without **new** operator will throw a compile time errors. For example:

// Error. Initialize an array without size

int[] array = new int[];

// Error. Initialize an array without new keyword

int[] array1;

array1 = { 1, 2, 3, 4, 5 };

## **C# Array Types**

In c#, we have a different type of arrays available, those are

* Single-Dimensional Arrays
* [Multi-Dimensional Arrays](https://www.tutlane.com/tutorial/csharp/csharp-multidimensional-arrays)
* [Jagged Arrays](https://www.tutlane.com/tutorial/csharp/csharp-jagged-arrays-with-examples)

## **C# Array Class**

In c#, we have a class called **Array** and it will act as a base class for all the arrays in common language runtime (CLR). The Array class provides a methods for creating, manipulating, searching and sorting an arrays.

For example, by using **Sort** or **Copy** methods of **Array** class we can sort the elements of an array and copy the elements of one array to another based on our requirements.

Following is the example of using an Array class to sort or filter or reverse array elements in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] array = new int[5] { 1, 4, 2, 3, 5 };

            Console.WriteLine("---Initial Array Elements---");

            foreach (int i in array)

            {

                Console.WriteLine(i);

            }

            Array.Sort(array);

            Console.WriteLine("---Elements After Sort---");

            foreach (int i in array)

            {

                Console.WriteLine(i);

            }

            Array.Reverse(array);

            Console.WriteLine("---Elements After Reverse---");

            foreach (int i in array)

            {

                Console.WriteLine(i);

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

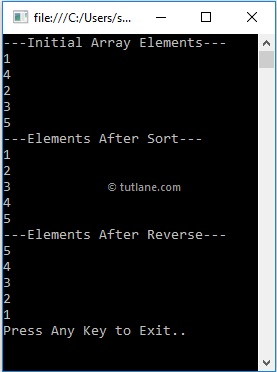
        }

    }

}

If you observe above example, we sorting and changing the order of array elements using **Sort** and **Reverse** methods of an **Array** class.

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



# C# Multidimensional Arrays

As discussed in previous chapter, [Arrays in C#](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) will support multi-dimensional arrays.  In c#, a multidimensional array is an array which contains a more than one dimension to represent the elements in tabular format like rows and columns.

In c#, multidimensional arrays can support either two or three dimensional series. To create multidimensional arrays, we need to use **comma** (**,**) separator inside the square brackets.

## **C# Multi-Dimensional Array Declaration**

In c#, **Multidimensional Arrays** can be declared by specifying the data type of an elements followed by the square brackets **[]** with **comma** (**,**) separator. Following are the examples of creating two or three dimensional arrays in c# programming language.

// Two Dimensional Array

int[,] arr = new int[4, 2];

// Three Dimensional Array

int[, ,] arr1 = new int[4, 2, 3];

## **C# Multidimensional Array Example**

Following is the example of using a multidimensional arrays in c# programming language to represent the elements in an array with multiple dimensions.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            // Two Dimensional Array

            int[,] array2D = new int[3, 2] { { 4, 5 }, { 5, 0 }, { 3, 1 } };

            // Three Dimensional Array

            int[, ,] array3D = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10, 11, 12 } } };

            Console.WriteLine("---Two Dimensional Array Elements---");

            for (int i = 0; i < 3; i++)

            {

                for (int j = 0; j < 2; j++)

                {

                    Console.WriteLine("a[{0},{1}] = {2}", i, j, array2D[i, j]);

                }

            }

            Console.WriteLine("---Three Dimensional Array Elements---");

            for (int i = 0; i < 2; i++)

            {

                for (int j = 0; j < 2; j++)

                {

                    for (int k = 0; k < 3; k++)

                    {

                        Console.WriteLine("a[{0},{1},{2}] = {3}", i, j, k, array3D[i, j, k]);

                    }

                }

            }

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

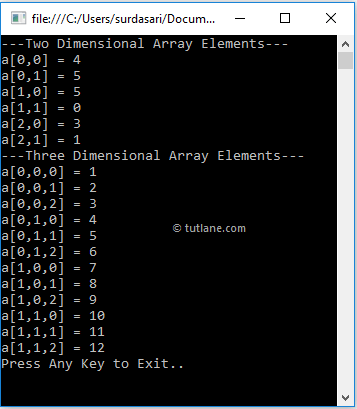
        }

    }

}

If you observe above example, we created a two and three dimensional arrays and getting those array values by using [for loop in c#](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples).

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



[2, 2, 3]

{ [0] [1]

[0] { { 1, 2, 3 }, { 4, 5, 6 } },

[1] { { 7, 8, 9 }, { 10, 11, 12 } }

}

Example: 0 1 0 =4

0 1 1 =5

# C# Jagged Arrays with Examples

In c#, **Jagged Array** is an array whose elements are arrays with different dimensions and sizes. Sometimes jagged array called as “array of arrays” and it can store arrays instead of particular data type value.

In c#, a jagged array can be initialized with two square brackets **[][]**. The first square bracket will specify the size of an array and the second one will specify the dimension of array which is going to be stored as a value.

Following are the examples of creating jagged arrays in c# programming language with [single](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) and [multidimensional](https://www.tutlane.com/tutorial/csharp/csharp-multidimensional-arrays) arrays.

// Jagged Array with Single Dimensional Array

int[][] jarray = new int[2][];

// Jagged Array with Two Dimensional Array

int[][,] jarray1 = new int[3][,];

If you observe above examples, first array (**jarray**) is allowed to store **2** elements of [single dimensional arrays](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) and second array (**jarray1**) is allowed to store 3 elements of [multidimensional arrays](https://www.tutlane.com/tutorial/csharp/csharp-multidimensional-arrays).

## **C# Jagged Array Initialization**

In c#, we can initialize an [arrays](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) upon declaration. Following are the different ways of declaring and initializing a jagged arrays in c# programming language.

// Jagged Array with Single Dimensional Array

int[][] jarray = new int[3][];

jarray[0] = new int[5] { 1, 2, 3, 4, 5 };

jarray[1] = new int[3] { 10, 20, 30 };

jarray[2] = new int[] { 12, 50, 60, 70, 32 };

// Jagged Array with Two Dimensional Array

int[][,] jarray1 = new int[3][,];

jarray1[0] = new int[2, 2] { { 15, 24 }, { 43, 54 } };

jarray1[1] = new int[,] { { 11, 12 }, { 13, 14 }, { 25, 26 } };

jarray1[2] = new int[4, 3];

// Initializing an Array on Declaration

int[][] jarray2 = new int[][]

{

new int[]{1,2,3,4,5},

new int[]{98, 56, 45},

new int[]{32}

};

## **C# Access Jagged Array Elements**

In c#, we can access the values of a jagged arrays by using row index and column index values.

Following is the example of accessing an elements from jagged arrays in c# programming language based on our requirements.

int i = jarray[0][2]; //3 (0th element 2nd (3rd) position

int j = jarray[2][1]; //50 (2nd element 1st (2nd position))

int k = jarray1[0][1, 1]; //54

int l = jarray1[1][2, 1]; //26

# C# Passing Arrays as Arguments

In c#, [arrays](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) are the [reference types](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples) so we can pass [arrays](https://www.tutlane.com/tutorial/csharp/csharp-arrays-with-examples) as an arguments to the [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) parameters and we can modify the values of an array elements inside of method based on our requirements.

# C# Classes and Objects with Examples

In c#, **Classes and Objects** are interrelated. The **class** in c# is nothing but a collection of various data members (fields, properties, etc.) and member functions. The **object** in c# is an instance of a **class** to access the defined properties and methods.

## **Declaring a Class in C#**

In c#, classes are declared by using class keyword. Following is the declaration of class in c# programming language.

public class users {

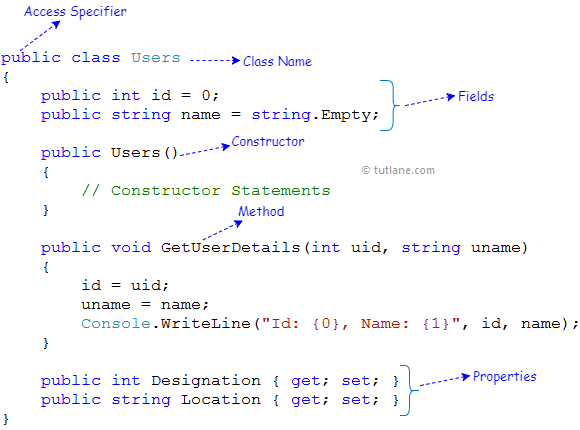
// Properties, Methods, Events, etc.

}

If you observe above syntax, we defined a class “**users**” using class keyword with public [access modifier](https://www.tutlane.com/tutorial/csharp/csharp-access-modifiers-public-private-protected-internal). Here, the public [access specifier](https://www.tutlane.com/tutorial/csharp/csharp-access-modifiers-public-private-protected-internal#divcpblm) will allow the users to create an objects for this class and inside of the body class, we can create a required [fields](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples), [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set), [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) and [events](https://www.tutlane.com/tutorial/csharp/csharp-events) to use it in our applications.

Now we will see how to create a class in c# programming language with example.

Following is the detailed description of various data members which we used in above c# class example.



## **C# Class Members**

As discussed, a **class** can contain multiple data members in c# programming language. Following table lists a different type of data members that can be used in c# classes.

| **Member** | **Description** |
| --- | --- |
| [Fields](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) | Variables of the class |
| [Methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) | Computations and actions that can be performed by the class |
| [Properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) | Actions associated with reading and writing named properties of the class |
| [Events](https://www.tutlane.com/tutorial/csharp/csharp-events) | Notifications that can be generated by the class |
| [Constructors](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples) | Actions required to initialize instances of the class or the class itself |
| [Operators](https://www.tutlane.com/tutorial/csharp/csharp-operators-arithmetic-relational-logical-assignment-precedence) | Conversions and expression operators supported by the class |
| [Constants](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword) | Constant values associated with the class |
| Indexers | Actions associated with indexing instances of the class like an array |
| Finalizers | Actions to perform before instances of the class are permanently discarded |
| Types | Nested types declared by the class |

## **C# Object**

In c#, **Object** is an instance of a **class** and that can be used to access the data members and member functions of a **class**.

# C# Constructors with Examples

In c#, **Constructor** is a method which will invoke automatically whenever an instance of [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) or **struct** is created.  The constructor will have a same name as the [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) or **struct**and it useful to initialize and set a default values for the data members of the new object.

In case, if we create a class without having any constructor, then the compiler will automatically create a one default constructor for that class. So, there is always one constructor will exist in every class.

In c#, a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) can contain more than one constructor with different type of arguments and the constructors will never return anything, so we don’t need to use any return type, not even **void** while defining the constructor method in [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples).

## **C# Constructor Types**

In c#, we have a different type of constructors available, those are

* [Default Constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr)
* [Parameterized Constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcspzcst)
* [Copy Constructor](https://www.tutlane.com/tutorial/csharp/csharp-copy-constructor-with-examples)
* [Static Constructor](https://www.tutlane.com/tutorial/csharp/csharp-static-constructor-with-examples)
* [Private Constructor](https://www.tutlane.com/tutorial/csharp/csharp-private-constructor-with-examples)

## **C# Default Constructor**

In c#, if we create a constructor without having any parameters, then we will call it as **default constructor** and the every instance of class will be initialized without any parameter values.

Following is the example of defining the default constructor in c# programming language.

using System;

namespace Tutlane

{

    class User

    {

        public string name, location;

        // Default Constructor

        public User()

        {

            name = "Suresh Dasari";

            location = "Hyderabad";

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // The constructor will be called automatically once the instance of class created

            User user = new User();

            Console.WriteLine(user.name);

            Console.WriteLine(user.location);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

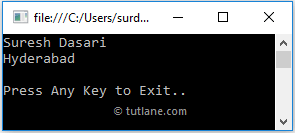
        }

    }

}

If you observe above example, we created a class called “**User**” and the constructor method “**User()**” without having any parameters. When we create an instance of our class (**User**), then automatically our constructor method will be called.

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



## **C# Parameterized Constructor**

In c#, if we create a constructor with at least one parameter, then we will call it as **parameterized constructor** and the every instance of class will be initialized with parameter values.

Following is the example of defining the parameterized constructor in c# programming language.

using System;

namespace Tutlane

{

    class User

    {

        public string name, location;

        // Parameterized Constructor

        public User(string a, string b)

        {

            name = a;

            location = b;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // The constructor will be called automatically once the instance of class created

            User user = new User("Suresh Dasari", "Hyderabad");

            Console.WriteLine(user.name);

            Console.WriteLine(user.location);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

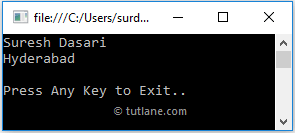
        }

    }

}

If you observe above example, we created a class called “**User**” and the constructor method “**User(string, string)**” with parameters. When we create an instance of our class (**User**) with required parameters, then automatically our constructor method will be called.

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



If you observe above result, our constructor method has called automatically and initialized the parameter values after creating an instance of our class with required parameters.

## **C# Constructor Overloading**

In c#, we can **overload** the constructor by creating another constructor with same method name but with different parameters.

Following is the example of implementing a constructor overloading in c# programming language.

using System;

namespace Tutlane

{

    class User

    {

        public string name, location;

        // Default Constructor

        public User() {

            name = "Suresh Dasari";

            location = "Hyderabad";

        }

        // Parameterized Constructor

        public User(string a, string b)

        {

            name = a;

            location = b;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User user = new User(); // Default Constructor will be called

            User user1 = new User("Rohini Alavala", "Guntur"); // Parameterized Constructor will be called

            Console.WriteLine(user.name + ", " + user.location);

            Console.WriteLine(user1.name + ", " + user1.location);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

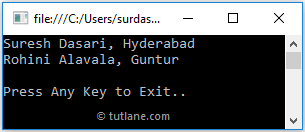
        }

    }

}

If you observe above example, we created a class called “**User**” and overloaded a constructor “**User()**” by creating another constructor “**User(string, string)**” with same name but with different parameters.

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



## **C# Constructor Chaining**

In c#, **Constructor Chaining** is an approach to invoke one constructor from another constructor. To achieve constructor chaining we need to use this keyword after our constructor definition.

Following is the example of implementing a **constructor chaining** in c# programming language.

using System;

namespace Tutlane

{

    class User

    {

        public User()

        {

            Console.Write("Hi, ");

        }

        public User(string a): this()

        {

            Console.Write(a);

        }

        public User(string a, string b): this("welcome")

        {

            Console.Write(a + " " + b);

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User user1 = new User(" to", "tutlane");

            Console.WriteLine();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

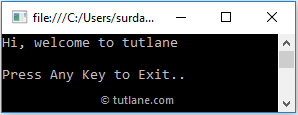
        }

    }

}

If you observe above example, we created a different constructors with different parameters and we are calling one constructor from another constructor using thiskeyword.

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



If you observe above result, we are able to call a one constructor from another constructor to achieve constructor chaining in c# programming language.

# C# Copy Constructor with Examples

In c#, **Copy Constructor** is a [parameterized constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcspzcst) which contains a parameter of same class type. The copy constructor in c# is useful whenever we want to initialize a new instance to the values of an existing instance.

In simple words, we can say copy constructor is a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples) which copies a data of one object into another object. Generally, c# won’t provide a copy constructor for objects but we can implement ourselves based on our requirements.

## **C# Copy Constructor Syntax**

Following is the syntax of defining a copy constructor in c# programming language.

class User

{

    // Parameterized Constructor

    public User(string a, string b)

    {

        // your code

    }

    // Copy Constructor

    public User(User user)

    {

        // your code

    }

}

If you observe the above syntax, we created a copy constructor with a parameter of same class type and it help us to initialize a new instance to the values of an existing instance.

## **C# Copy Constructor Example**

Following is the example of creating a copy constructor to initialize a new instance to the values of an existing instance in c# programming language.

using System;

namespace Tutlane

{

    class User

    {

        public string name, location;

        // Parameterized Constructor

        public User(string a, string b)

        {

            name = a;

            location = b;

        }

        // Copy Constructor

        public User(User user)

        {

            name = user.name;

            location = user.location;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // User object with Parameterized constructor

            User user = new User("Suresh Dasari", "Hyderabad");

            // Another User object (user1) by copying user details

            User user1 = new User(user);

            user1.name = "Rohini Alavala";

            user1.location = "Guntur";

            Console.WriteLine(user.name + ", " + user.location);

            Console.WriteLine(user1.name + ", " + user1.location);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

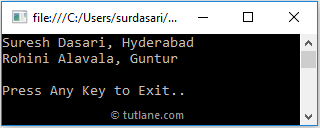
        }

    }

}

If you observe above example, we created an instance of copy constructor (**user1**) and using an instance of **user** object as a parameter type. So the properties of **user** object will be send to **user1** object and we are changing the property values of **user1** object but those will not effect the **user** object property values.

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



If you observe above result, we initialized a new instance to the values of an existing instance but those changes not effected the existing instance values.

# C# Static Constructor with Examples

# C# Static Constructor with Examples

In c#, **Static Constructor** is used to perform a particular action only once throughout the application. If we declare a [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples) as **static**, then it will be invoked only once irrespective of number of class instances and it will be called automatically before the first instance is created.

Generally, in c# the static constructor will not accept any access modifiers and parameters. In simple words we can say it’s a parameter less.

Following are the properties of static constructor in c# programming language.

* Static constructor in c# won’t accept any parameters and access modifiers.
* The static constructor will invoke automatically, whenever we create a first instance of class.
* The static constructor will be invoked by CLR so we don’t have a control on static constructor execution order in c#.
* In c#, only one static constructor is allowed to create.

## **C# Static Constructor Syntax**

Following is the syntax of defining a static constructor in c# programming language.

class User

{

   // Static Constructor

   static User()

   {

      // Your Custom Code

   }

}

If you observe above syntax, we created a static constructor without having any parameters and access specifiers.

## **C# Static Constructor Example**

Following is the example of creating a static constructor in c# programming language to invoke the particular action only once throughout the program.

using System;

namespace Tutlane

{

    class User

    {

        // Static Constructor

        static User()

        {

            Console.WriteLine("I am Static Constructor");

        }

        // Default Constructor

        public User()

        {

            Console.WriteLine("I am Default Constructor");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // Both Static and Default constructors will invoke for first instance

            User user = new User();

            // Only Default constructor will invoke

            User user1 = new User();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

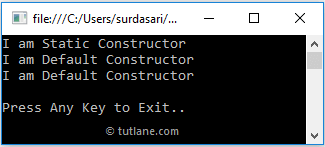
        }

    }

}

If you observe above example, we created a **static constructor** and [default constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr). Here the static constructor will be invoked only once for the first instance of class.

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



If you observe above result, for the first instance of class both static and default constructors execute and for the second instance of class only [default constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr) has executed.

# C# Private Constructor with Examples

In c#, **Private Constructor** is a special instance constructor and it is used in a classes that contains only **static** members. If a class contains one or more private constructors and no public constructors, then the other classes are not allowed to create an instance for that particular class except nested classes.

## **C# Private Constructor Syntax**

Following is the syntax of defining a private constructor in c# programming language.

class User

{

  // Private Constructor

  private User()

  {

     // Your Custom Code

  }

}

If you observe above syntax, we created a private constructor without having any parameters that means it will prevent an automatic generation of [default constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr). In case, if we didn’t use any **access modifier** to define [constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples), then by default it will treat it as a **private**.

## **C# Private Constructor Example**

Following is the example of creating a private constructor in c# programming language to prevent other [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) to create an instance of particular [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples).

using System;

namespace Tutlane

{

    class User

    {

        // private Constructor

        private User()

        {

            Console.WriteLine("I am Private Constructor");

        }

        public static string name, location;

        // Default Constructor

        public User(string a, string b)

        {

            name = a;

            location = b;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // The following comment line will throw an error because constructor is inaccessible

            //User user = new User();

            // Only Default constructor with parameters will invoke

            User user1 = new User("Suresh Dasari", "Hyderabad");

            Console.WriteLine(User.name + ", " + User.location);

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

        }

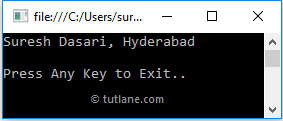
    }

}

If you observe above example, we created a class with **private constructor** and [default constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr) with parameters. If you uncomment the commented line (User user = newUser();), then it will throw an error because the constructor is a private so it won’t allow you to create an instance for that class.

Here we are accessing class properties directly with class name because those are **static** properties so it won’t allow you to access with instance name.

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



# C# Destructor with Examples

In c#, **Destructor** is a special method of a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) and it is used in a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) to destroy the object or instances of [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples). The destructor in c# will invoke automatically whenever the class instances become unreachable.

Following are the properties of destructor in c# programming language.

* In c#, destructors can be used only in [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) and a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) can contain only one destructor.
* The destructor in [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) can be represented by using **tilde** (**~**) operator
* The destructor in c# won’t accept any parameters and access modifiers.
* The destructor will invoke automatically, whenever an instance of [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) is no longer needed.
* The destructor automatically invoked by garbage collector whenever the [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) objects that are no longer needed in application.

## **C# Destructor Syntax**

Following is the syntax of defining a destructor in c# programming language.

  class User

    {

        // Destructor

        ~User()

        {

            // your code

        }

    }

If you observe above syntax, we created a destructor with same [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) name using **tilde** (~) operator. Here we need to remember that destructor name must always same as [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) name in c# programming language.

## **C# Destructor Example**

Following is the example of using destructor in c# programming language to destruct the unused objects of class.

using System;

namespace Tutlane

{

    class User

    {

        public User()

        {

            Console.WriteLine("An Instance of class created");

        }

        // Destructor

        ~User()

        {

            Console.WriteLine("An Instance of class destroyed");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Details();

            GC.Collect();

            Console.ReadLine();

        }

        public static void Details()

        {

            // Created instance of class

            User user = new User();

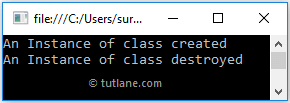
        }

    }

}

If you observe above example, we created a class with [default constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples#divcsdfcstr) and **destructor**. Here we created an instance of class “**User**” in **Details()** method and whenever the **Details** function execution is done, then the garbage collector (**GC**) automatically will invoke a destructor in **User** class to clear the object of class.

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



# C# this Keyword

In c#, **this** keyword is used to refer the current instance of class and by using this keyword we can pass current instance of class as a parameter to the other methods.

In case, if class contains a parameters and variables with same name, then this keyword is useful to distinguish between the parameters and variables.

We can also use this keyword to declare indexers and to specify the instance variable in the parameter list of an extension method.

## In c#, we should not use this keyword to refer **static** field or method and also it cannot used in **static classes**.

## **C# this Keyword Syntax**

Following is the syntax of using this keyword in c# programming language.

this.instance\_variable

If you observe above syntax, this is a keyword and **instance\_variable** is an instance variable name.

## **C# this keyword Example**

Following is the example of using this keyword in c# programming language to refer the class variables and parameters of same name and use this keyword to send instance of [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) to the method of another [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples).

using System;

namespace Tutlane

{

    class User

    {

        public string name, location;

        public long marks = 470;

        public User(string name, string location)

        {

            // Use this to distinguish between parameters and variables

            this.name = name;

            this.location = location;

        }

        public void GetUserDetails()

        {

            Console.WriteLine("Name: {0}", name);

            Console.WriteLine("Location: {0}", location);

            // Passing a class instance to the method using this

            Console.WriteLine("Marks: {0}", Exams.GetPercentage(this));

        }

    }

    class Exams

    {

        public static double GetPercentage(User u)

        {

            double i = ((double)470 / 600) \* 100;

            return (i);

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User u = new User("Suresh Dasari", "Hyderabad");

            u.GetUserDetails();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

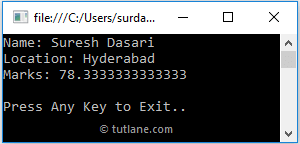
        }

    }

}

If you observe above example, we used this keyword to distinguish between class variables and parameters of same name and used this keyword to send instance of class (**User**) to the method of another class.

When we run above c# program, we will get the result like as shown below.



# C# Static Keyword

In c#, **static** is a keyword or a modifier which is used to make a class or methods or a variable properties as not instantiable that means we cannot instantiate the items which we declared with a static modifier.

The **static** members which we declared can be accessed directly with a type name. Suppose if we apply a static modifier to a class property or to a method or variable, then we can access those static members directly with a class name, instead of creating an object of a class to access those properties.

## **C# Static Variables**

Following is the example of defining a class with static properties and those can be access directly with a type instead of specific object name.

class User

{

public static string name, location;

public static int age;

}

If you observe above example, we defined a variables with static keyword and we can access those variables directly with a type name like **User.name** or **User.location** and **User.age**.

Following is the example of accessing the variables directly with a type name in c# programming language.

Console.WriteLine(User.name);

Console.WriteLine(User.location);

Console.WriteLine(User.age);

Generally, in c# the instance of class will contain a separate copy of all instance fields so the memory consumption will increase automatically, but if we use static modifier there is only one copy of each field so automatically the memory will be managed efficiently.

In c#, we can use static modifier with [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples), [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples), properties, [constructors](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples), [operators](https://www.tutlane.com/tutorial/csharp/csharp-operators-arithmetic-relational-logical-assignment-precedence), fields and with events but it cannot be used with **indexers**, **finalizers** or types other than [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples).

## **C# Static Keyword Example**

Following is the example of creating a class by including both static and non-static variables & methods. Here we can access non-static variables and methods by creating an instance of the class, but it’s not possible for us to access the static fields with an instance of class so the static variables and methods can be accessed directly with the class name.

## **C# Static Keyword Example**

Following is the example of creating a class by including both static and non-static variables & methods. Here we can access non-static variables and methods by creating an instance of the class, but it’s not possible for us to access the static fields with an instance of class so the static variables and methods can be accessed directly with the class name.

using System;

namespace Tutlane

{

class User

{

// Static Variables

public static string name, location;

//Non Static Variable

public int age;

// Non Static Method

public void Details()

{

Console.WriteLine("Non Static Method");

}

// Static Method

public static void Details1()

{

Console.WriteLine("Static Method");

}

}

class Program

{

static void Main(string[] args)

{

User u = new User();

u.age = 32;

u.Details();

User.name = "Suresh Dasari";

User.location = "Hyderabad";

Console.WriteLine("Name: {0}, Location: {1}, Age: {2}", User.name, User.location, u.age);

User.Details1();

Console.WriteLine("\nPress Enter Key to Exit..");

Console.ReadLine();

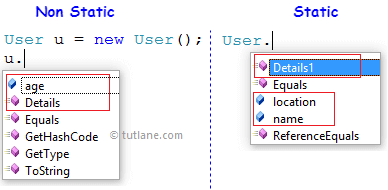
}

}

}

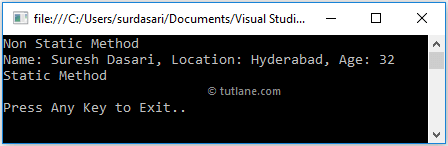
If you observe above example, we created a class called “**User**” with **static** and **non-static** variables & methods. Here we are accessing **non-static** variables and methods with an instance of **User** class and **static** fields & methods are able to access directly with the class name (**User**).

Following diagram will illustrate more details about how **static** and **non-static** variables & methods can be accessed in our c# application.



If you observe above diagram, it clearly says that **non-static** fields and methods can be accessed only with an instance of class and the **static** fields & methods can be accessed directly with the class name.

When we run above c# program, we will get the result like as shown below.



# C# Static Class with Examples

In c#, a **static class** can be created by using static modifier and the static class can contain only static members.

Generally, the **static class** is same as **non-static** **class**, but only difference is the **static class** cannot be instantiated. Suppose if we apply static modifier to a class, then we should not use the **new** keyword to create a variable of the class type.

Another difference is the **static class** will contain only [static](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword) members, but the **non-static class** can contain both [static](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword) and non-static members.

## **C# Static Class Syntax**

In c#, we can create a **static class** by applying static keyword to the class like as shown below.

static class sample

 {

     //static data members

    //static methods

 }

If you observe above syntax, to create a static class called “**sample**”, we applied static keyword to the class type. The methods and data members which we are going to implement in sample class must be a **static**.

In c#, we can access a members of static class directly with the class name. For example, we have a static class called “**User**” with a method “**Details()**” that we can access like **User.Details()**.

## **C# Static Class Example**

Following is the example of defining a **static class** to access data members and member functions without creating an instance of class in c# programming language.

using System;

namespace Tutlane

{

    static class User

    {

        // Static Variables

        public static string name;

        public static string location;

        public static int age;

        // Static Method

        public static void Details()

        {

            Console.WriteLine("Static Method");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            User.name = "Suresh Dasari";

            User.location = "Hyderabad";

            User.age = 32;

            Console.WriteLine("Name: {0}", User.name);

            Console.WriteLine("Location: {0}", User.location);

            Console.WriteLine("Age: {0}", User.age);

            User.Details();

            Console.WriteLine("\nPress Enter Key to Exit..");

            Console.ReadLine();

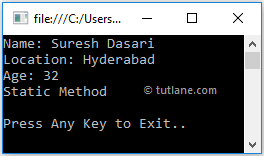
        }

    }

}

If you observe above example, we are accessing **static** class members and functions directly with the class name because we cannot instantiate the **static class**.

When we execute above c# program, we will get the result like ass shown below.



## **C# Static Class Features**

Following are the main features of static class in c# programming language.

* The static class in c# will contain only **static** members.
* In c#, the static classes cannot be instantiated.
* C# static classes are **sealed**, therefore it cannot be inherited.
* The static classes in c# will not contain [instance constructors](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples).

As discussed in previous article [static keyword in c#](https://www.tutlane.com/tutorial/csharp/csharp-static-keyword), we can use **static** members in **non-static**[classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) such as normal [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples). For normal [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) we can create an instance of class using **new** keyword to access non-static members and functions but it cannot access the **static** members and [functions](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples).