**Differences Between Var and Dynamic in C#:**

Now, let us summarizes the differences between var and dynamic in C#. The differences are as follows:

**Var in C#**

1. var is known as a statically typed variable, meaning that the data type of these variables is identified at compile time, which is done based on the type of value with which these variables are initialized.
2. var in C# was introduced as part of C# 3.0.
3. In the case of var, the variable’s data type is identified by the compiler at the compilation time only.
4. In the case of var, it is mandatory to initialize the variable at the time of its declaration so that the compiler comes to know the variable’s data type according to the right-hand side value assigned to it.
5. It will throw an error if the variable does not initialize at the time of its declaration.
6. We will get intelligence support in the visual studio.
7. Var cannot be used to declare method parameters and method return type in C#. It can only be used as a local variable declaration inside a function.
8. Var is early bounded. This means the compiler decides the type of variable declared at compile time.

**Dynamic in C#**

1. Dynamic is known as a dynamically typed variable which means that the data type of these variables is identified at runtime, which is done based on the type of value that these variables are initialized with.
2. Dynamic in C# was introduced in C#4.0.
3. In the case of dynamic, the data type of variable is identified by the CLR at run time.
4. In the case of dynamic, it is not mandatory to initialize the variable at the time of its declaration.
5. It will not throw an error if the variable does not initialize at the time of its declaration.
6. We will not get any intelligence support in the visual studio.
7. Dynamic can be used to declare method parameters and method return type in C#. It can also be used as a local variable declaration inside a function.
8. Dynamic is Late Bounded. This means the CLR decides the type of variable declared at runtime time.

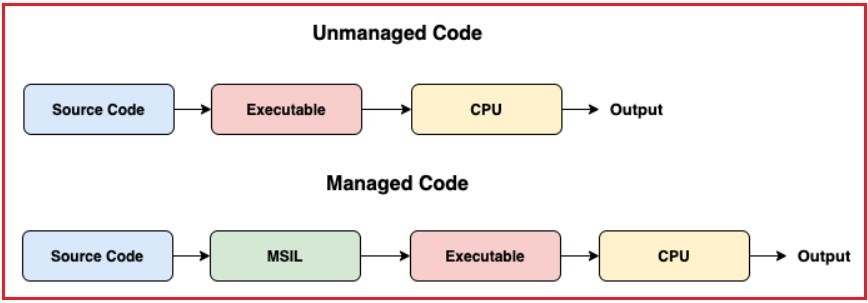
## **Ref Vs Out**

|  |  |
| --- | --- |
| Ref | Out |
| The parameter or argument must be initialized first before it is passed to ref. | It is not compulsory to initialize a parameter or argument before it is passed to an out. |
| It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method. | A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method. |
| Passing a parameter value by Ref is useful when the called method is also needed to modify the passed parameter. | Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method. |
| It is not compulsory to initialize a parameter value before using it in a calling method. | A parameter value must be initialized within the calling method before its use. |
| When we use REF, data can be passed bi-directionally. | When we use OUT data is passed only in a unidirectional way (from the called method to the caller method). |
| Both ref and out are treated differently at run time and they are treated the same at compile time. |  |
| Properties are not variables, therefore they cannot be passed as an out or ref parameter. |  |

**What exactly is the Managed and Unmanaged code in .NET Framework?**

The codes which run under the complete control of CLR are called Managed Code in .NET Framework. These kinds of code (Managed code) are run by the .NET Runtime Environment i.e. CLR. If the .NET Framework is not installed or if the .NET Runtime Environment i.e. CLR is not available, then these kinds of codes are not going to be executed. CLR will provide all the facilities and features of .NET to the managed code execution like Language Interoperability, Automatic Memory Management, Exception Handling Mechanism, Code Access Security, Garbage Collection, etc. In this case, the source code is compiled in the intermediate language known as IL or MSIL, or CIL.

On the other hand, Skype, PowerPoint, and Microsoft Excel do not require .NET Runtime Environment, they run under their own environment. So, in short, the code (EXE, DLL) which does not run under the control of CLR is called Unmanaged Code. CLR will not provide any facilities and features of .NET to the Unmanaged Code like Language Interoperability, Automatic Memory Management, Exception Handling Mechanism, Code Access Security, Garbage Collection, etc. In this case, the source code directly compiles into native languages.



ADVERTISEMENT

**Note:** Managed code is the code that is managed by the CLR (Common Language Runtime) in .NET Framework. Whereas Unmanaged code is the code that is directly executed by the operating system.

**What are the advantages of using Managed Code?**

1. It improves the security of the application like it will check whether the current has access to assembly or not and whether it is safe to execute the assembly by the Operating System or not.
2. Whenever an object is not used by the application, then the Garbage Collector automatically destroys the memory for unused objects.
3. It will support default exception handling.

**What are the disadvantages of Managed Code?**

1. The main disadvantage of managed code in the .NET Framework is that we are not allowed to allocate memory directly, or we cannot get low-level access to the CPU architecture. This is going to be managed by the CLR of the .NET Framework.

**What are the advantages of using Unmanaged Code?**

1. It provides the low-level access to the programmer.
2. It also provides direct access to the hardware of the machine.
3. The code is a little bit faster than managed code.
4. We can run the Unmanaged Code under any environment and platform.

**What are the disadvantages of Unmanaged Code?**

1. It does not provide security to the application.
2. Due to the access to memory allocation, issues related to memory leakage might have occurred.
3. No automatic garbage collection.
4. It will not support default exception handling, the developer needs to take care of that things.

##### ****What is .NET Framework?****

According to Microsoft, .NET Framework is a software development framework for building and running applications on Windows. The .NET Framework is part of the .NET platform, a collection of technologies for building apps for Linux, macOS, Windows, iOS, Android, and more.

##### ****What are Indexers in C#?****

The indexer in C# is a property of a class that allows us to access a member variable of a class using the features of an array. That means the Indexers in C# are the members of a class and if we define indexers in a class then the class behaves like a virtual array.

So, the Indexers in C# allow instances of a class to be indexed just like arrays. The indexed value can be set or retrieved without explicitly specifying a type or instance member. If this is not clear at the moment then don’t worry, we will understand this concept with examples.

###### **C# Definition based on the above features:**

**C# is just a Simple, Secure, Robust, Portable, Platform-Independent, Architectural Neutral, Multithreaded, Automatic Memory Management, Object-Oriented Programming Language with a strong type Exception Handling mechanism for developing different kinds of applications such as Web, Windows Form, Console, Web Services, Mobile Apps, etc. which can be run on different Operating Systems such as Windows, Linus, and Mac.**

##### ****Features of C#****

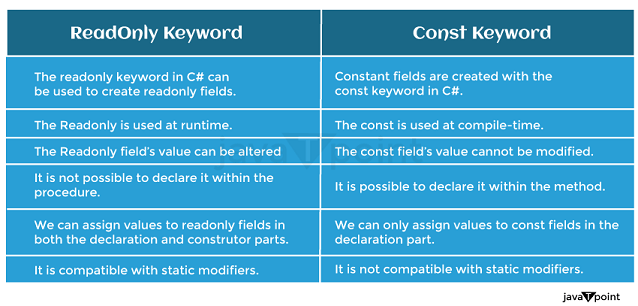
Although C# constructs closely follow traditional high-level languages, C and C++ and being an object-oriented programming languages. It has a strong resemblance with Java, it has numerous strong programming features that make it endearing to a number of programmers worldwide. Following is the list of a few important features of C# −

1. **Simple**: C# is a simple language in the sense that it provides a structured approach (to break the problem into parts), a rich set of library functions, data types, etc.
2. **Modern Programming Language**: C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable, and robust applications.
3. **Object-Oriented**: C# is an object-oriented programming language. OOPs makes development and maintenance easier whereas in Procedure-oriented programming language it is not easy to manage if code grows as the project size grows.
4. **Type-Safe**: C# type safe code can only access the memory location that it has permission to execute. Therefore, it improves the security of the program.
5. **Interoperability**: The interoperability process enables the C# programs to do almost anything that a native C++ application can do.
6. **Scalable and Updateable:** C# is an automatic scalable and updateable programming language. For updating our application, we delete the old files and update them with new ones.
7. **Component Oriented:** C# is a component-oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.
8. **Structured Programming Language:** C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.
9. **Rich Library:** C# provides a lot of inbuilt functions that make development fast.
10. **Fast Speed:** The compilation and execution time of the C# language is fast.

##### ****Types of Applications Developed using C#:****

With the help of the C# programming language, we can develop different types of secured and robust applications:

1. **Window applications**
2. **Web applications**
3. **Distributed applications**
4. **Web service applications**
5. **Database applications**
6. **Mobile Applications, etc**



|  |  |
| --- | --- |
| **ReadOnly Keyword** | **Const Keyword** |
| **In C#, readonly fields can be created using readonly keyword** | **In C#, constant fields are created using const keyword.** |
| **ReadOnly is a runtime constant.** | **Const is a compile time constant.** |
| **The value of readonly field can be changed.** | **The value of the const field can not be changed.** |
| **It cannot be declared inside the method.** | **It can be declared inside the method.** |
| **In readonly fields, we can assign values in declaration and in the constructor part.** | **In const fields, we can only assign values in declaration part.** |
| **It can be used with static modifiers.** | **It cannot be used with static modifiers.** |

# **Difference between String and StringBuilder in C#**

[C#](https://www.tutorialsteacher.com/articles?category=csharp)

By TutorialsTeacher

 28 Mar 2023

In C#, both [string](https://www.tutorialsteacher.com/csharp/csharp-string) and [StringBuilder](https://www.tutorialsteacher.com/csharp/csharp-stringbuilder) are used to represent text. However, there is one key difference between them.

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

For example, the following create a new string object when you concatenate a value to it.

Example: String

 Copy

string greeting = "Hello World!";

greeting += " from Tutorials Teacher."; // creates a new string object

In contrast, StringBuilder is a mutable type. It means that you can modify its value without creating a new object each time.

Example: StringBuilder

 Copy

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

sb.Append("from Tutorials Teacher."); //appends to the same object

The StringBuilder performs faster than the string if you modify a string value multiple times. If you modify a string value more than five times then you should consider using the StringBuilder than a string.

##### ****Convert.ToString and ToString Method in C#****

Both these methods are used to convert a value to a string. The difference is **Convert.ToString()** method handles null whereas the **ToString()** doesn’t handle null in C#.

In C#, a dictionary is a collection of key-value pairs, while a list is a group of items. Both are generic collections that store data and have random access data structures. However, they have different performance characteristics when it comes to inserting and searching data:

* Dictionary

Dictionaries are slower to insert data because they need to calculate a hash, but are faster to search because of that hash. Dictionaries are a good choice if you need fast retrieval based on keys or unique key-value pairs.

* List

Lists are faster to insert data because they don't need to calculate a hash, but are slower to search because they need to look at every element. Lists are a good choice if you need an ordered collection, sequential access, or indexing operations. Lists are also helpful for storing large amounts of data

##### ****What is the difference between an Array and an Array List in C#?****

The ArrayList collection in C# is very much similar to the Arrays data type. The major difference between them is the dynamic nature of the non-generic collection ArrayList. For arrays, we need to define the size i.e. the number of elements that the array can hold at the time of array declaration. But in the case of the ArrayList collection in C#, this does not need to be done beforehand. Elements can be added or removed from the Array List collection at any point in time.

This is one of the frequently asked interview questions in C#. So let us discuss the difference between an array and an ArrayList.

###### **Array:**

1. Fixed Length
2. Cannot insert it into the middle
3. Cannot delete from middle
4. It is type-safe, so we can store only similar types of data based on the data type.
5. Boxing and Unboxing are not required.

###### **ArrayList:**

1. Variable Length
2. Can insert an element into the middle of the collection
3. Can delete elements from the middle of the collection
4. It is not type-safe, so we can store any type of data.
5. Boxing and Unboxing are required as it is operated on the object data type.

**What is a Collection in C#?**

The **Collections in C#** are a set of predefined classes that are present in the **System.Collections** namespace that provides greater capabilities and functionalities than the traditional arrays. The collections in C# are reusable, more powerful, and more efficient and most importantly they have been designed and tested to ensure quality and performance.

So in simple words, we can say a Collection in C# is a dynamic array**.** That means the collections in C# have the capability of storing multiple values but with the following features.

1. Size can be increased dynamically.
2. We can insert an element into the middle of a collection.
3. It also provides the facility to remove or delete elements from the middle of a collection.

The collections in C# are classes that represent a group of objects. With the help of C# Collections, we can perform different types of operations on objects such as Store, Update, Delete, Retrieve, Search, and Sort objects, etc. In short, all the data structure work can be performed by collections in C#. That means Collections standardize the way in which the objects are handled by our program.

##### ****What are Delegates in C#?****

In simple words, we can say that the delegates in C# are the Type-Safe Function Pointer. It means they hold the reference of a method or function and then call that method for executio

**Rules of using Delegates in C#:**

1. A delegate in C# is a user-defined type and hence before invoking a method using a delegate, we must have to define that delegate first.
2. The signature of the delegate must match the signature of the method, the delegate points to otherwise we will get a compiler error. This is the reason why delegates are called type-safe function pointers.

**What are the Types of Delegates in C#?**

The Delegates in C# are classified into two types as

1. **Single Cast Delegate**
2. **Multicast Delegate**

ADVERTISEMENT

If a delegate is used for invoking a single method, then it is called a single cast delegate or unicast delegate. In other words, we can say that the delegates that represent only a single function are known as single cast delegates.

If a delegate is used for invoking multiple methods, then it is known as the multicast delegate. Or the delegates that represent more than one function are called Multicast delegates.

**Where do we use Delegates in C#?**

Delegates are used in the following cases:

1. Event Handlers
2. Callbacks
3. Passing Methods as Method Parameters
4. LINQ
5. Multithreading

**How many ways we can call a method in C#?**

In C#, we can call a method that is defined in a class in two ways. They are as follows:

1. We can call the method using the object of the class if it is a non-static method or we can call the method through the class name if it is a static method.
2. We can also call a method in C# by using delegates. Calling a C# method using delegate will be faster in execution as compared to the first process i.e. either by using an object or by using the class name.