# **Reflection in C#**

جهت کار کردن و دسترسی به محتوای یک اسمبلی کاربرد دارد. می توانید به صورت dynamic یک instance از یک type بسازید یا متد های یک نوع را فراخونی کنید یا prop ها را مقدار دهی کنید و به آن ها دسترسی داشته باشید.

کاربرد اصلی دسترسی به محتوای assembly است. محتوای public private و ...

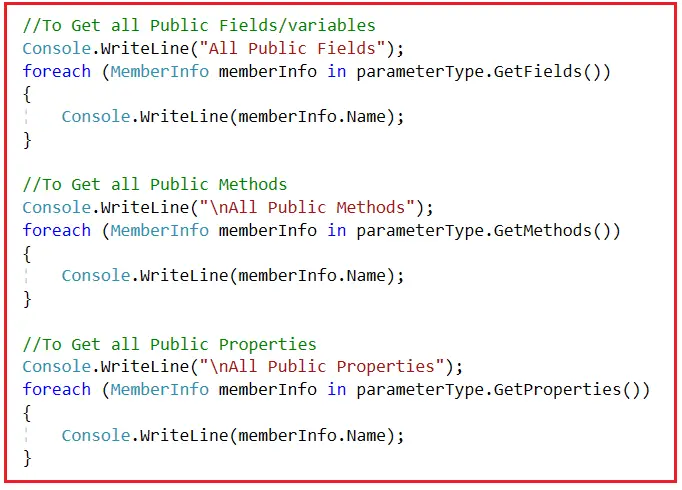
**مثال**

زمانی که در ویژوال استدیو از یک type یک object ایجاد می کنیم و دکمه دات را می زنیم تمامی گزینه های که intelligence می دهد تمامی prop method filed و ... امکان پذیر است به دلیل وجود reflection

**Sample**

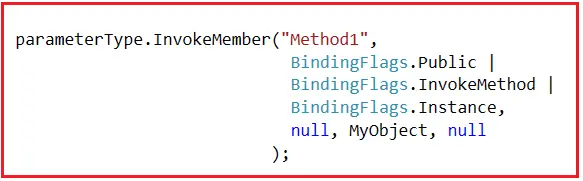
در ابتدا نیاز است که اسمبلی خود را Load کنیم با استفاده از Assembly.Loadfile(path) جهت دریافت reference مربوط به فایل assembly سپس نیاز است که class refence ها را دریافت کنیم از اسمبلی خود با استفاه از متد GetType سپس نیاز است که یک instance از کلاس خود ایجاد کنیم با استفاده از Activator.CreateInstance(myType) بعد از دریافت object می توانیم نوع Class را دریافت کنیم با استفاده از instance.GetType()

برای دسترسی به تمامی public member ها لازم است از متد GetMembers() استفاده کنیم و برای متد ها GetMethods() و فیلد ها GetFilelds() استفاده کنیم و پروپرتی ها GetPRoperties() .



**How to Invoke Methods Dynamically Using Reflection in C#?**

برای invoke کردن یک متد می توان از روش زیر استفاده کرد.



**Method Parameters**

1. **name: The string containing the name of the constructor, method, property, or field member to invoke. In our case it is Method1.**
2. **invokeAttr: A bitmask comprised of one or more System.Reflection.BindingFlags that specify how the search is conducted. The access can be one of the BindingFlags such as Public, NonPublic, Private, InvokeMethod, GetField, and so on. The type of lookup need not be specified. If the type of lookup is omitted, BindingFlags.Public | BindingFlags.Instance | BindingFlags.Static is used.**
3. **binder: An object that defines a set of properties and enables binding, which can involve the selection of an overloaded method, coercion of argument types, and invocation of a member through reflection. -or- A null reference to use the System.Type.DefaultBinder. Note that explicitly defining a System.Reflection.Binder objects may be required for successfully invoking method overloads with variable arguments. Here, we are passing a null value.**
4. **target: The object on which to invoke the specified member. In our example, the object is MyObject.**
5. **args: An array containing the arguments to pass to the member to invoke. As our method does not take any arguments, we pass null here.**

**Sample**

**//Step2: Get the type**

**//Get the Assembly Reference**

**var MyAssembly = Assembly.LoadFile(@"D:\Projects\ReflectionDemo\SomeClassLibrary\bin\Debug\SomeClassLibrary.dll");**

**//Get the Class Reference**

**var MyType = MyAssembly.GetType("SomeClassLibrary.Class1");**

**//Create an instance of the type**

**dynamic MyObject = Activator.CreateInstance(MyType);**

**//Get the Type of the class**

**Type parameterType = MyObject.GetType();**

**//Step3: Browse the Metadata**

**//To Get all Public Fields/variables**

**Console.WriteLine("All Public Members");**

**foreach (MemberInfo memberInfo in parameterType.GetMembers())**

**{**

**Console.WriteLine(memberInfo.Name);**

**}**

**Console.WriteLine("\nInvoking Method1");**

**parameterType.InvokeMember("Method1",**

**BindingFlags.Public |**

**BindingFlags.InvokeMethod |**

**BindingFlags.Instance,**

**null, MyObject, null**

**);**

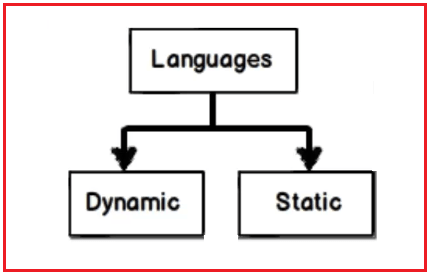
**Real Time Uses**

برای دسترسی به متا دیتا ها - در unit test – Late binding برای فایل های که در compile به آن ها دسترسی نداریم - زمانی که 2 پیاده سازی داریم از یک interface در فایل های مختلف پس لازم است که به صورت dynamic مشخص کنیم از کدام پیاده سازی استفاده شود.

# **Dynamic Type in C#**

برای جلوگیری از compile time checking کاربرد دارد و در زمان runtime عملیات چکینگ را انجام می دهد.

**Types of Programming Languages:**



Dynamic programing language (dynamic)

Strongly typed programing language (static)

**Strongly typed programing languages** زبان های هستند که **Data type checking** در زمان کامپایل رخ می دهد.

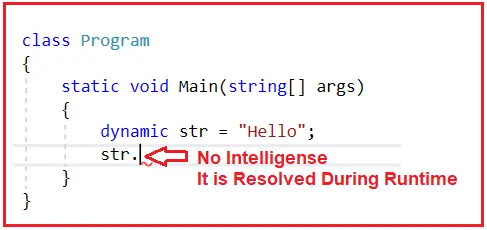
مثل دادن عدد به نوع String که خطای کامپایل می هد.

**Dynamic type** زبان های هستند که Data type checking در زمان run time انجام می شود.

مواقعی است که نوع داده را در زمان compile نمی دانیم پس نیاز است که data type checking را به زمان run time منتقل کنیم.

کلمه کلیدی dynamic کار ترکیبی انجام میدهد به این معنی که در زمان کامپایل type checking را انجام نمی دهد اما در زمان run time عملیات بررسی نوع داده رخ می دهد.

مثال زیر در زمان کامپایل داده را چک نمی کند اما در زمان اجرا بررسی می کند نوع آن را.



فهمیدن نوع داده در زمان اجرا با reflection امکان پذیر است.

**dynamic** str = "Hello";

Console.WriteLine**(**str.GetType**())**;

خروجی کنسول system string است.

**Static binding – early binding به معنای عملیات بایند در زمان کامپایل است.**

**نکته:** تبدیل نوع داده static به نوع داده dynamic در سی شارپ به صورت خود کار اتفاق می افتد emplicitly.

**int** int1 = 50;

**dynamic** dynamic1 = int1; //Explicit cast not required

Console.WriteLine**(**$"int1 = {int1} & dynamic1 = {dynamic1}"**)**;

//Convert from dynamic to int

**dynamic** dynamic2 = 100;

**int** int2 = dynamic2; //Explicit cast not required

**dynamic with complex data type**

**dynamic** dynamicStudent = new Student**()**

**{**

Name = "Anurag",

Barnch = "CSE",

Roll = 1001

نکته مهم در تبدیل نوع داده ها احتمال data loss است.

Dynamic parameter

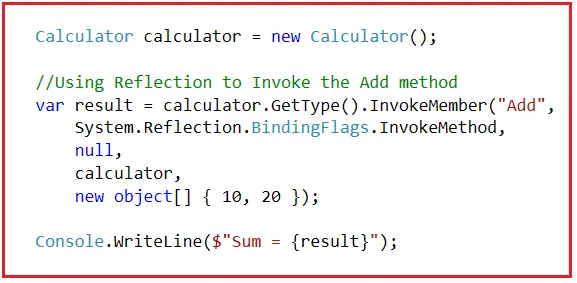
**public** **static** **void** DisplayValue**(dynamic** val**)**

**{**

Console.WriteLine**(**val**)**;

**}**

یکی از مزایای آن استفاده برای نگه داری json است



فراخوانی متد با reflection و ارسال 2 پارامتر به آن

**public** **class** Calculator

**{**

**public** **int** Add**(int** number1, **int** number2**)**

**{**

**return** number1 + number2;

**}{**

**با استفاده از Reflection نیاز به نوشتن کد زیادی برای دریافت متدی از یک کلاس هستیم اما با dynamic به سادگی می توان انجام داد به دلیل این که در زمان کامپایل type check انجام نمی شود**

**dynamic** calculator = new Calculator**()**;

var result = calculator.Add**(**10, 20**)**;عدم خطا چون کامپایل چک نمی کند و همچنین دارای متد ادد در نوع کلاس هستیم که زمان اجرا بررسی می وشد

# **Var Keyword in C#**

**Implicitly Typed Local Variables**

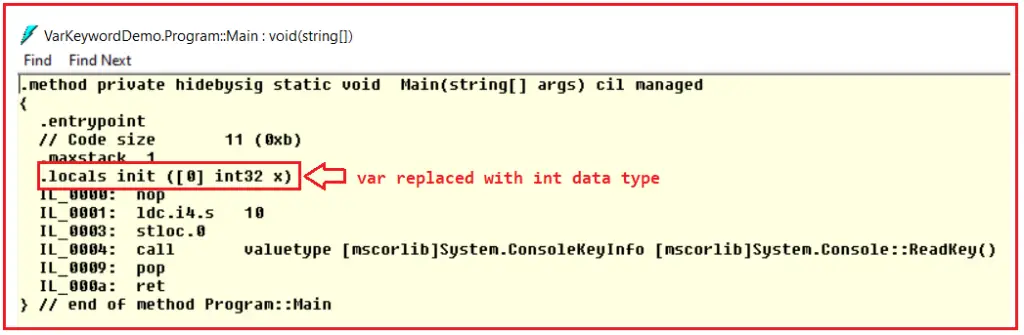
با استفاده از Var نوع variable به صورت خودکار توسط compiler تشخیص داده می شود بر اساس assign سمت راست خود.

**Explicit declaration or direct declaration**

Int a = 12;

در زمان کامپایل تبدیل نوع var به نوع داده صحیح اتفاق می افتد.

اگر فایل اسمبلی را در idlasm باز کنید می توانید ببینید که نوع داده var به نوع داده دلخواه شما تبدیل شده است.



نکته type checking , type safety در زمان compile با استفاده از var رخ می دهد.

**یکی از مزایا : کوتاه شدن کد مثلا کلاسی با نام طولانی دارید که نیازی نیست پشت متغییر اسم کلاس و جلوش هم اسم بنویسید.**

**مزایای دیگر : استفاده در anonymous type و در linq query در حلقه ها for – foreach**

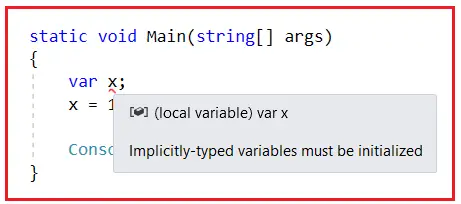
**نکته : declaration – initialization برای Var باید در یک statement رخ دهد.**

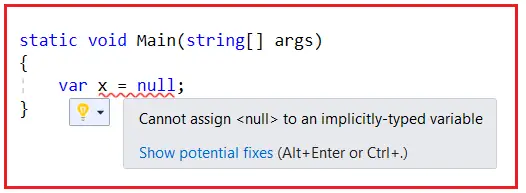
**امکان null برای مقدار دهی اولیه Var وجود ندارد.**

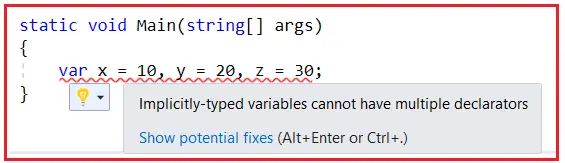
**امکان مقدار دهی چند متغییر در یک statement وجود ندراد**

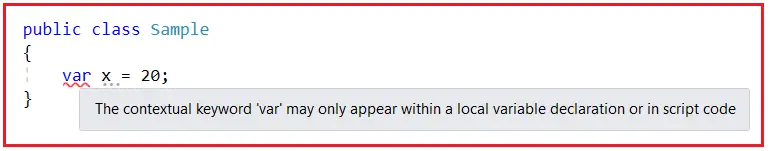
**قابلیت استفاده به عنوان filed را ندارد.**

1. **With the var keyword, we will get better performance as boxing and unboxing are not required.**

****

****

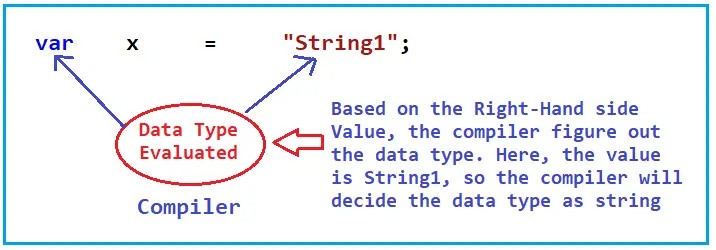
****

****

# **Var vs Dynamic in C#**

Varearly binding

Dynamic late binding



**dynamic** d = "String1"; //Late Bounded

**int** Len2 = d.Length;

در زمان runtime متغییر d با استفاده از reflection سعی میکند که پروپرتی lengh را فراخوانی کند اگر وجود داشته باشد اجرا می شود و اگر نه Exception می دهد.

اگر همان property را با استفاده از l کوچک در dynamic بنویسیم در زمان compile خطا نمی دهد اما در زمان run سبب بروز خطا می شود چون CaseSensetive است و L بزرگ را می شناسد نه کوچک.

**نکته**

باید توجه داشت که var فقط در محدوده **local variable inside function** کاربرد دارد اما **نمی** **تواند** **property** باشد یعنی class member

Dynamic از reflection در پشت صحنه استفاده می کند.

**Can we Change the Type of value after being assigned in Var and Dynamic in C#?**

امکان تغییر نوع داده متغییر Var بعد از Assign کردن وجود **ندارد.**

**در dynamic امکان تغییر نوع داده بعد Assign وجود دارد.**

**dynamic** str = "Somevalue";

str = 10; //str type changed from string to int type at Runtime

**Can we use var and dynamic as a function’s return type or parameter in C#?**

**امکان استفاده از var بع عنوان return type یک متد وجود ندارد.**

**امکان استفاده از dynamic به عنوان Return Type متد وجود دارد**

**static dynamic SomeMethod(dynamic x, dynamic y)**

**{**

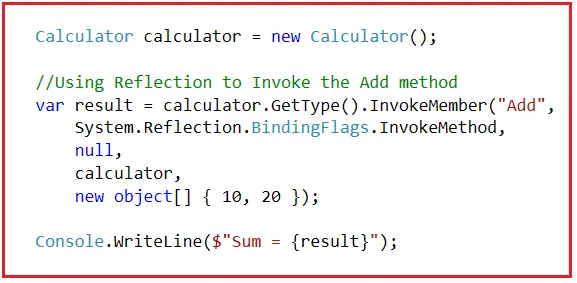
**return x + y;**

**}**

# **Dynamic vs Reflection in C#**

با استفاده از dynamic نوشتن کد های reflection بسیار ساده تر و خوانا تر است.

مثال Reflection



کد قبلی با استفاده از dynamic

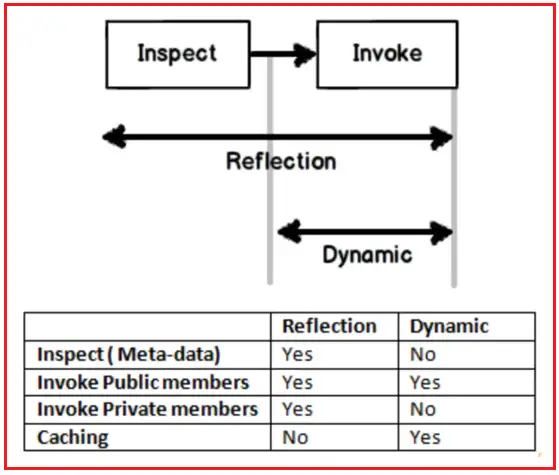
//Using dynamic to Invoke the Add method

**dynamic** calculator = new Calculator**()**;

var result = calculator.Add**(**10, 20**)**;

با استفاده از dynamic نوشتن کد ها سریع تر راحت تر و خوانا تر است نسب به reflection

سناریو های استفاده از dynamic , reflection



نکته:

Dynamic به صورت internally از reflection استفاده می کند و 2 ابزار برای این کار دارد اول استفاده از meta data برای دسترسی به method property filed های مربوط به یک Assembly و ابزار دیگر برای فراخوانی method property ,filed

**Notice**

**تفاوت مهم در metadata است که در حالت reflection به آن دسترسی داریم و اگر متا دیتا خواستیم استفاده کنیم اما اگر فقط فراخوانی method filed prop را می خواهیم از dynamic استفاده کنیم.**

عدم دسترسی dynamic به metadata.

هردو قابلیت استفاده از اعضای public را دارند.

اعضای private فقط توسط Rflection قابل استفاده هستند.

کش کردن با dynamic امکان پذیره ولی با reflection خیر.

**Volatile Keyword in C#**

یکی از ناشناخته ترین کلمات کلید در زبان سی شارپ است.

زمانی که در حال توسعه اپلیکیشن های multi Thread هستید پر کاربرد سات.

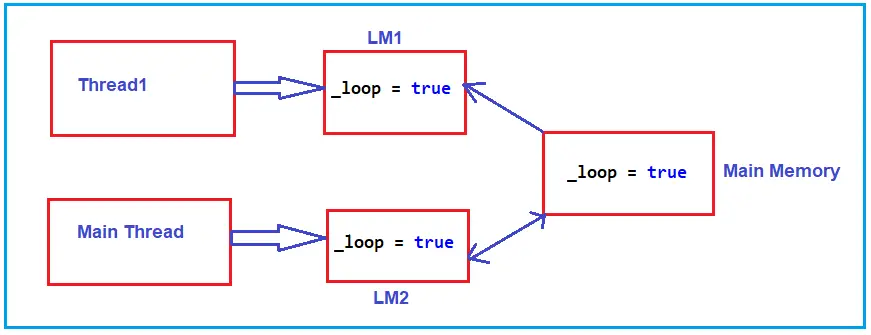
طبق مستندات ماکروسافت کلمه کلیدی volatile مشخص می کند که یک field ممکن است توسط چند thread که به صورت هم زمان درحال اجرا هستند تغییر کند.

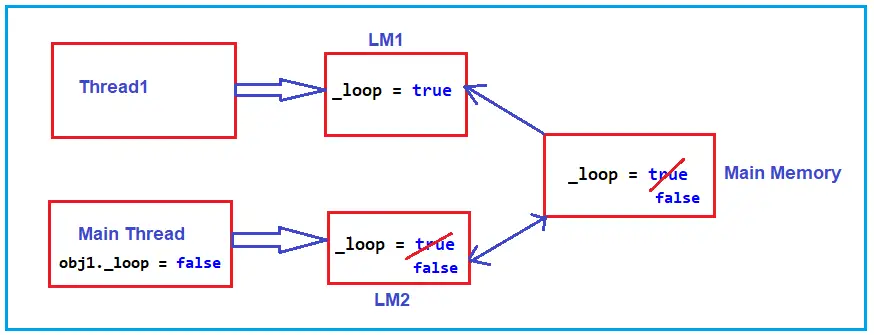
نکته :

**کامپایلر ، runtime system و سخت افزار ممکن است به دلایل performance مکان حافظه را تغییر دهند اما فیلد های که از نوع volatile هستند از انواع خاصی از بهینه سازی مستثنی هستند.**

**نکته مهم**

**هر thrad دارای Local Memory مخصوص به خود است که ممکنه شما در Thread mean یک متغییری را تغیر داده باشید اما در سایر thread همان مقدار قبلی باشد که سبب اشتباه می شود.**

****

****

**مشکل این است که Local Memory , main memory با یک دیگر Sync نیستند.**

**چگونه می توان کاری کرد زمانی که thread1 تفییر در field انجام داد مقدار در thred قبلی نیز تعییر کند.**

**راه حل**

**private volatile bool \_loop = true;**

**while loop accesses this \_loop variable, first, it will go and sync this local memory \_loop variable data with the Main memory \_loop variable data and then it will execute the loop**

**زمانی که آپ چند نخی داری و مقداری را تغییر می دهی می خواهی سایر ترد ها هم مطع شود و sync باشند از این کلمه کلیدی استفاده کنید. پس حالتی که اگر تغییر در متغییر داشتیم حافظه اصلی تغییر کند و همه ترد ها با خبر باشند از مقدار جدید.**

**می گوید که مقدار نباید کش شود.**

**The reason why the code is giving an infinite loop in release mode but works normally in debug mode is because of the way the CPU caches variables. In debug mode, the debugger forces the CPU to flush the cache every time the value of a variable changes. This ensures that all threads have the most up-to-date version of the variable. However, in release mode, the CPU is free to cache variables as it sees fit. This means that it is possible for one thread to see a stale value of a variable, even if another thread has updated the value.  
In the code, the \_loop variable is declared as a bool. By default, bool variables are not volatile. This means that the CPU is free to cache the value of the variable. If the CPU caches a stale value of the \_loop variable, then the while loop will never terminate.**

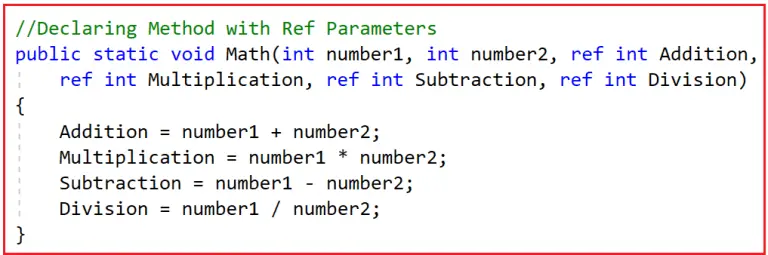
**To fix this problem, we can mark the \_loop variable as volatile. This will tell the CPU that the value of the variable can change at any time, and that it should not be cached.**

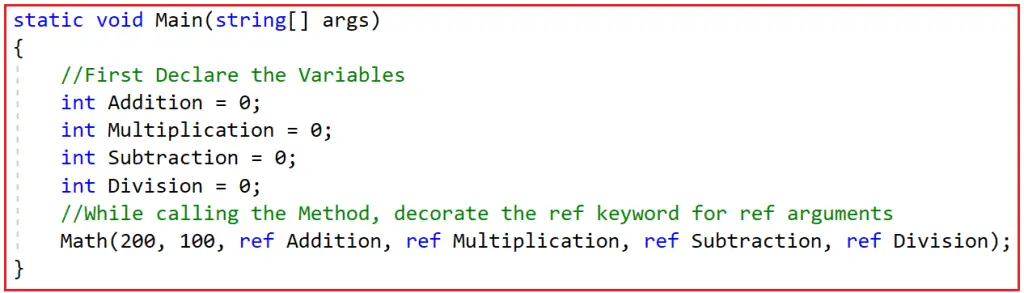
# **Ref vs Out in C#**

هر دوی این کلمات کلیدی برای پاس دادن argument به متد کاربرد دارند به عنوان reference type .

به صورت پیشفرض arguments به صورت by value به متد ارسال می شوند اما با استفاده از کلمات ref , out امکان ارسال به صورت reference نیز وجود دارد.

در مثال زیر متغییر ها را به صورت by referece ارسال می کنیم این متغییر ها که از نوع int هستند به صورت پیش فرض by value ارسال می شوند یعنی یک کپی از داده به متد می رود که اگر تغییر کند داده اصلی دستخوش تغییر نمی شود اما در مثال به صورت reference ارسال می کنیم که با تغییر داده اصلی نیز تغییر کند.





زمانی که با ref متغییری که از نوع Value است را ارسال می کنیم در واقع Reference آن را ارسال کردیم که شامل pointer به داده اصلی که با تغییر آن داده اصلی نیز تغییر می کند.

**Example using Out Parameter**

کلمه کلیدی out را می توان برای بازنویسی کد قبلی استفاده کرد و همان پاسخ را گرفت اما سوال اصلی این است که چه تفاوتی میان ref و out است؟

**زمانی که یک argument را با استفاده از کلمه کلیدی out ارسال می کنیم متد باید به صورت mandatory  مقدار آن را در داخل خود تغییر دهد. اگر تغییر نکند compile error می دهد.**

زمانی که از Ref استفاده می کنید اجباری است که متغییر از قبل initialize شده باشد اما در out نیازی نیست که از قبل initialize شده باشد چون در داخل متد init و تغییر می کند

Ref برای ارسال داده دوطرفه کاربرد دارد و out برای ارسال داده به صورت یک طرفه کاربرد یعنی return کاربرد دارد.

Ref برای زمانی که انتظار ارسال مقدار و تغییر و دریافت مقداری را داریم.

**Out برای زمانی که مقدار نمی فرستیم و انتظار مقدار داریم. به دلیل init شدن توسط متد نیازی به init قبل ارسال ندارد و می توان در پارامتر متد آن را ارسال کرد بدون تعریف متغییر مجزا.**

//Use of out in C#

Add**(**10, 20, out **int** Number**)**;

**Named Parameters in C#**

راهکاری برای مچ کردن یک آرگومان با نام آن است به جای این که بر اساس مکان پارامتر تصمیم گیری انجام شود و این named parameters می تواند در method , indexer , ctor , delegate کاربرد داشته باشد.

فراخوانی متد با روش named parameter

employee.AddEmployeeInfo**(IsPermanent:** true, **Name:** "Pranaya", **Department:"**IT" **)**;

**امکان استفاده ی ترکیبی از simple argument , named parameter وجود دارد فقط به این صورت که اول باید fixed argument جاگذاری شود و سپس named parameter در غیر این صورت خطا کامپایل می دهد.**

در حالتی که optional پارامتر داریم نیز می توانیم از named برای مشخص کردن آن های که می خواهیم استفاده کنیم.