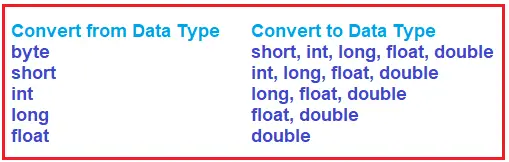
# Type Casting in C#

# تبدیل یک نوع داده به نوع داده دیگر تنها در صورتی که با یک دیگر سازگار باشند.

# Implicit Type Casting

# تبدیل یک نوع داده کوچکتر به نوع داده بزرگتر که به صورت خودکار توسط کامپایلر انجام می شود. نوع داده ها باهم سازگار باشند.



**Explicit Type Casting**

در این حالت تبدیل نوع داده امکان از بین رفتن بخشی از داده وجود دارد و ما باید به طور مستقیم تبدیل داده خود را اعلام کنید مثلا اگر نوع داده double را به int تبدیل کنیم قسمت اعشار از بین می رود یا ممکن است نوع داده ی رشته ی نامربوط را به نوع داده int بخواهیم تبدیل کنیم که سبب بروز خطا می شود.

روش های تبدیل نوع داده صریح

(int)

Convert.ToInt32()

Int.Parse()

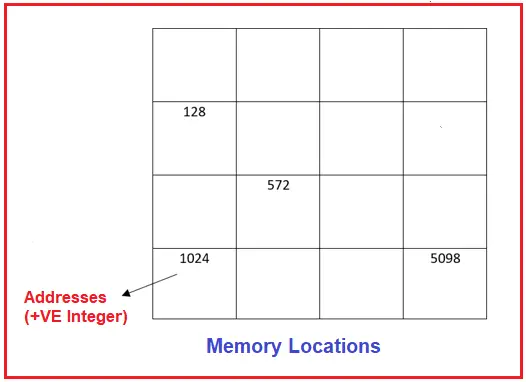
Int.tryParse()

امن ترین روش گزینه اخر است جون در صورت بروز خطا false بر میگرداند و در صورتی که موفق باشد مقدار خروجی را در متغییری به ما تحویل می دهد.

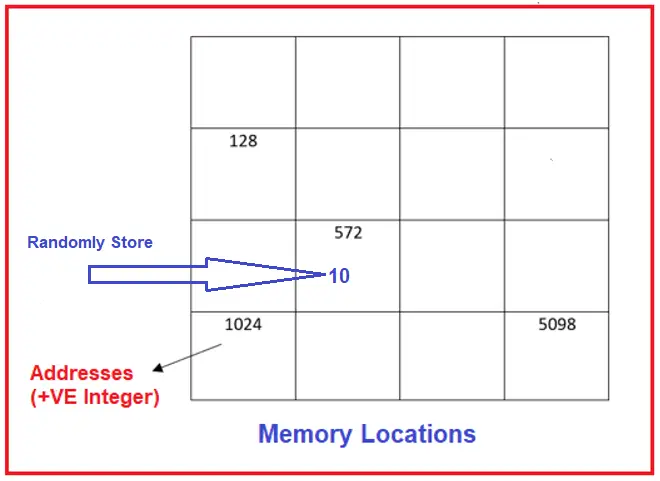
## **Variables in C# with Examples**

هر عملیاتی که در سیستم های نرم افزاری در حال انجام است به طور خلاصه بیانگر این است که بر روی داده های مثل شماره حساب بانکی مقدار موجودی و ... پردازش صورت می گیرد. پس بنابر این تمامی مقایر که روی آن ها پردازش صورت می گیرد باید در مکانی از حافظه وجود داشته باشند.

تمامی مکان های حافظه مانند صندلی های یک سالن تئاتر است که هر کدام دارای آدرس شناسایی منحصر به فرد خود است.



وقتی می خواهیم یک مقدار مثل عدد 10 را در حافظه ذخیره کنیم به صورت رندوم در یکی از خانه های حافظه که با شماره خانه محل آدرس آن مشخص است مقدار عدد ما ذخیره م شود



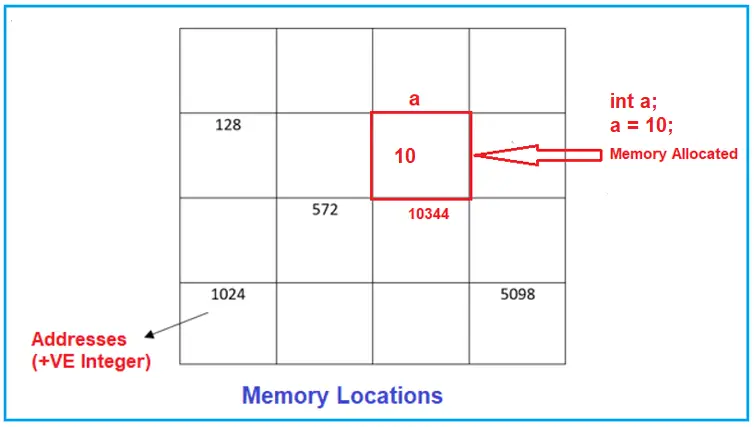
How To Access To Data

قبل از ذخیره کردن داده برای بازیابی آن باید آدرس محل حافظه که داده در آن ذخیره می شود را نگه داری کنیم.

How We Can Set Identify To Memory Location

می توانیم آدرس حافظه را در یک متغییر ذخیره کنیم

به طور مثل وقتی int a را داریم نوع داده را بیان کردیم و هویتش را با هر اسمی مشخص می کنیم مثلا a سپس به آن حافظه تخصیص داده می شود و با شناسه a محل حافظه آن مشحص می شود الان a نام محل حافظه به شماره ی 10344 است.



##### **What is a Variable in C# Language?**

نامی است که به یک خانه از حافظه داده شده است. هدف متعییر تخصیص نام به مکانی از حافظه است. کاربر متغییر را با نامش می شناسند و کامپایلر با داده آن را با آدرس حافظه.

نام متغییر باید

با letter یا underscore شروع شود

به صورت case sensitive است

می تواند با نوع داده خود constract شود

علائم خاص نمی پذیرد

##### **Types of Variables in a Class in C#**

Instance

Static

Constant

ReadOnly

نکته : اگر یک متغییر را در یک بلاک کد از نوع static تعریف کنیم نوع آن متغییر نیز از نوع static در نظر گرفته می شود.

using System;

namespace TypesOfVariables

{

internal class Program

{

static int x = 100; //Static Variable

int y = 200; //Non-Static or Instance Variable

static void Main(string[] args)

{

Console.WriteLine($"x value: {x}");

Console.WriteLine($"x value: {y}");

Console.Read();

}

}

}

قطعه کد بالا دارای خطا است به دلیل این که یک متغییر غیر استاتیک به نام y داریم که در یک متد از نوع static استفاد شده است باید این نکته را در نظر گرفت که تمامی متغییر های که از نوع non static هستند برای تخصیص حافظه به خود نیازمند ایجاد instance هستند به این معنی که با توجه به هر instance ساخته شده از خود به آن ها حافظه تعلق می گیرد در این مثال چون متد ما از نوع ststic است و نیازی به ساخت instance برای اجرا ندارد در آن نمی توان از متغییر non static استفاده کرد.

تنها در حالتی می شود که از کلاس program در این متد instance ساخته شود و به y مقداری تخصیص داده شود و سپس آن را چاپ کرد.

using System;

namespace TypesOfVariables

{

internal class Program

{

static int x = 100; //Static Variable

int y = 200; //Non-Static or Instance Variable

static void Main(string[] args)

{

Console.WriteLine($"x value: {x}");

Program obj = new Program();

Console.WriteLine($"y value: {obj.y}");

Console.Read();

}

}

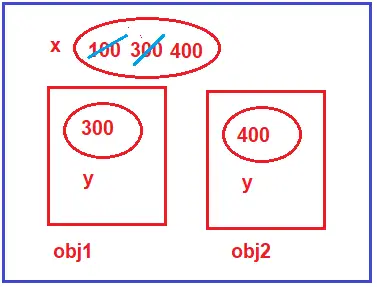
}

##### **When Static and Non-Static Variables are Initialized in C#?**

متغییر های استاتیک یک بر بلافاصله پس از شروع اجرای کد کلاس مقدار دهی می شوند و تا پایان عمر کلاس دارای همان مقدار می باشند اما متغییر های که از نوع غیر استاتیک هستند با هر ساخت نمونه از کلاس مقدار دهی می شوند

Can We Assign Value To Static Member With Constractor?

بله در هر با ساخت نمونه از کلاس می توانیم به متغییر استاتیک خود در سازنده مقداری را assign کنیم اما باید به این نکته توجه کرد که هر با مقداری دهی متغییر استاتیک باعث می شود که مقدار قبلی override شود به مقدار جدید.



###### **Instance/Non-Static Variables in C#**

1. **Scope of Instance Variable:** Throughout the class except in static methods.
2. **The lifetime of Instance Variable:** Until the object is available in the memory.

###### **Static Variables in C#**

1. **Scope of the Static Variable**: Throughout the class.
2. **The Lifetime of Static Variable**: Until the end of the program.

Constant Variable In C#

یک بار در موقعه ی ایجاد مقدار دهی می شود و از آن به بعد غیر قابل تغییر است اگر در موقع تعریف مقدار دهی نشود با خطای کامپایلر مواجه می شوید. به این دلیل که فقط یک بار ایجاد می شوند.

Const نیز نیازی به ساخت instance برای تخصیص مقدار به خود ندارد.

Const int a = 12;

ReadOnly

تشابه زیادی با متغییر های از نوع const دارد فقط با یک تفاوت که نیازی نیست در لحظه ی تعریف به آن مقدار دهید بلکه می توانید با constractor به آن مقدار دهید.

تقاوت با Const

Const یک مقدار ثابت برای کل کلاس ها است اما ReadOnly یک مقدار ثابت برای هر instance از کلاس است.

**Local Variables in C#**

1. **Scope of the Local Variables:** Within the block in which it is declared.
2. **The lifetime of the Local Variable:** Until the control leaves the block in which it is declared.

# ****Operators in C# with Examples****

# **Operator => + = ++ ,…**

# **Operand => 1 , 2, a ,v ,…**

**Types of Operators in C#**

**Arithmetic**

**Relational Or Comparison**

**Logical**

**Assignment**

**Bitwise**

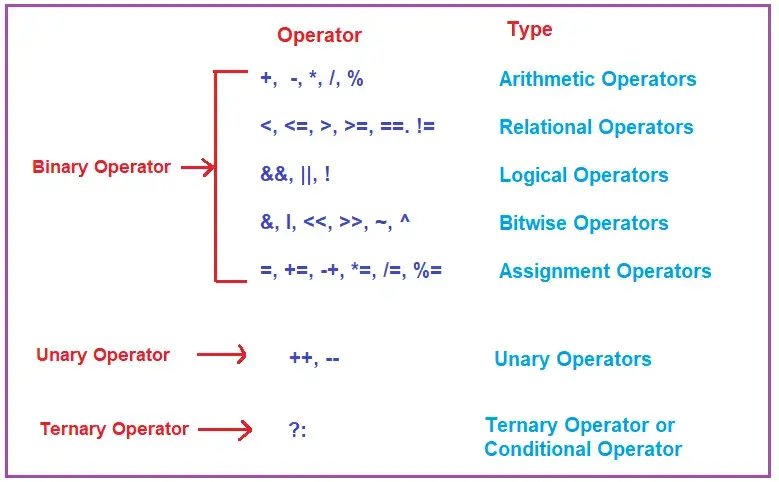
**Unary**

**Ternary**

**Conditional**

**با توجه به تعداد operand ها نیز تقسیم می شوند**

**Unary – binary – ternary**



##### **Arithmetic Operators in C#**

* **Addition Operator (+):**
* **Subtraction Operator (-):**
* **Multiplication Operator (\*):**
* **Division Operator (/):**
* **Modulus Operator (%):**

**Assignment Operators in C#**

* **Simple Assignment (=):**
* **Add Assignment (+=):**
* **Subtract Assignment (-=):**
* **Multiply Assignment (\*=):**
* **Division Assignment (/=):**
* **Modulus Assignment (%=):**

##### 

##### **Relational Operators in C#**

رابطه ای یا مقایسه ای نیز گفته می شود به این ها که معمولا به صورت boolen نتیجه را برمیگرداند.

* **Equal to (==):**
* **Not Equal to (!=):**
* **Less than (<):**
* **Less than or equal to (<=):**
* **Greater than (>):**
* **Greater than or Equal to (>=):**

**Logical Operators in C#**

* Logical OR (||)
* Logical AND (&&)
* Logical Not (!)

##### **Bitwise Operators in C#**

**Bitwise OR (|)**

عملگر بیتی | یا OR بر روی بیت های متناظر بین دو عملوند عملیات انجام می دهد اگر هر یک از بیت ها 1 باشد 1 می دهد و اگر نه 0 می دهد.

**int a=12, b=25;**  
**int result = a|b; //29**  
**How?**  
12 Binary Number: **00001100**  
25 Binary Number: **00011001**  
Bitwise OR operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00011101 (it is 29 in decimal)**

**Bitwise AND (&):**

اگر هر دو 1 باشد 1 می دهد و اگر هر دو 0 باشد 0 می دهد.

**int a=12, b=25;**  
**int result = a&b; //8**  
How?  
12 Binary Number: 00001100  
25 Binary Number: 00011001  
Bitwise AND operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00001000 (it is 8 in decimal)**

**Bitwise XOR (^):**

**اگر بیت ها یکی باشند مقدار 0 می دهد و اگر بیت ها یکی نباشند مقدار 1 می دهد.**

**int a=12, b=25;**  
**int result = a^b; //21**  
How?  
12 Binary Number: 00001100  
25 Binary Number: 00011001  
Bitwise AND operation between 12 and 25:  
**00001100**  
**00011001**  
**========**  
**00010101 (it is 21 in decimal)**

**Unary Operators in C#**

1. **Increment operators (++): Example: (++x, x++)**
2. **Decrement operators (–): Example: (–x, x–)**
3. **Post-Increment Operator**
4. **Pre-Increment Operator**

**Syntax: Variable++;**  
**Example: x++;**

**Syntax: ++Variable;**  
**Example: ++x;**

##### **Post Decrement Operators**

**Syntax: Variable–;**  
**Example: x–;**

**Syntax: –Variable;**  
**Example: —x;**

##### **Ternary Operator in C#**

**Syntax: Condition? first\_expression : second\_expression;**

# Goto Statement in C#

# برای انتقال کنترل برنامه به label کاربرد دارد

# Label در واقع یک شناسه مشخص است. توصیه می شود که استفاده نشود چون خوانایی را کاهش داده و پیچیدگی را افزایش می دهد.

# 

using System;

namespace JumpStatementDemo

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("1st Statement: ");

goto label1; //goto label label1:

Console.WriteLine("2nd Statement: ");

label1: //label:

Console.WriteLine("3rd Statement: ");

Console.WriteLine("End of Main Method Statement: ");

Console.ReadKey();

}

}

}

# اگر لیبل مورد نظر وجود نداشته باشد خطای کامپایلری مواجه می شوید.

# از goto برای خروج از حلقه ها نیز می توان استفاده کرد.

از goto برای انتقال کنترل برنامه به یک case خاص در switch نیز می توان استفاده کرد

نوشتن حلقه با goto

**int** count = 1;

label\_Loop:

Console.WriteLine**(**count**)**;

count++;

**if** **(**count **<**= 10**)**

**{**

**goto** label\_Loop;

**}**

Console.Write**(**"C#Tutorials "**)**;

Console.Write**(**"Welcome "**)**;

XYZ:

Console.Write**(**"X "**)**;

Console.Write**(**"Y "**)**;

**goto** ABC;

Console.Write**(**"Programming "**)**;

ABC:

Console.Write**(**"Hello1 "**)**;

Console.Write**(**"Hello2"**)**;

Console.ReadKey**()**;

**Output: C#Tutorials Welcome X Y Hello1 Hello2**

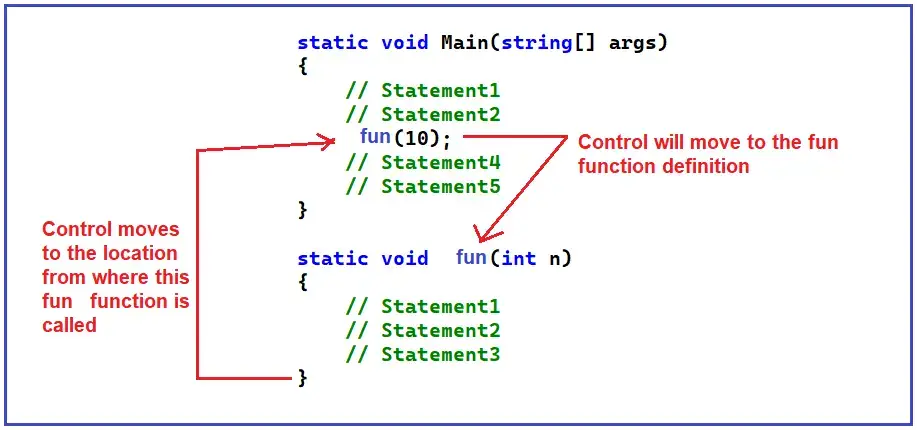
**اگر به یک برچسب برسیم که صدا نزیدم آن را کد های بعد از آن اجرا می شود و ارتباطی به عدم صدا کردن آن ندارد**

**در حلقه های goto عبارت های continue , break بی معنی است**

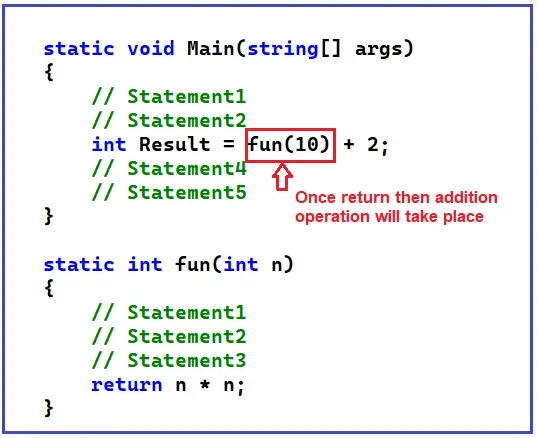
**برچسب ها به صورت caseSensetive هستند.**

### Recursion in C#

زمانی که متدی صدا زده می شود کنترل برنامه به آن متد رفته و پس از پایان عملیات مجدد کنترل برنامه به خط صدا زننده متد بازمی گردد.



اگر با مقدار بازگشتی از متد fun بخواهید کاری انجام دهید تنها زمانی قادر خواهید بود که متد مقداری را بازگردانده باشد.



نکته : اگر متدی a صدا کند متد b را و متد b صدا کند متد c را ترتیب انجام شدن کار ها اول اتمام متد c سپس b و در آخر a است.

##### **What does it mean by Recursive Function in C#?**

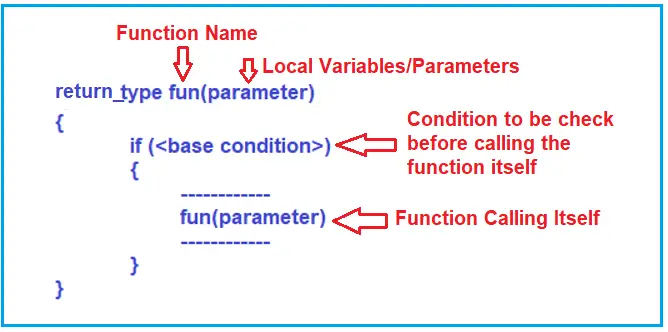
متدی که خودش را صدا می زند تا زمانی که یک شرط به نتیجه برسد و return کند.

برای حل مسائل recursive نیاز است که 2 شرط داشته باشیم.

اول مشکل باید به صورت recursive نوشته شود و متد خودش را صدا بزند

دوم باید شرطی برای پایان کار نیز داشته باشیم

زمانی که در متد recursive دارای متغییر محلی هستیم به اضای هر بار فراخوانی تابع متغییر ها مقدار های جدید را می گیرند و در stack مقدار نگه داشته می شود



متد های بازگشتی به صورت درختی trace می شوند.

نکته کار متد های بازگشتی تا پایان ریسدن ادامه دارد یعنی اولین صدا تا آخرین صدا زدن life cycle است.

از مزایای متد های بازگشتی حفط اطلاعات مربوط به فراخوانی تابع با بازگشت آن است.

معایب

کندی به دلیل stack overlapping

ممکن است سبب stack overflow شود

مکنن است سبب infinite loop شود

##### **How do you find the time complexity of a recursive function in C#?**

فرض کنید هر دستور در برنامه ما یک واحد زمانی برای اجرا نیاز دارد. حالا باید تکرار دفعاتی که متد بازگشتی با دستوری را اجرا می کند بشماریم در این مثال متد ما فقط مقداری را چاپ می کند بنابر این به ازای هر باز اجرا یک کار انجام میدهد یعنی رابطه ی دفعات تکرار برابر است با تعداد ورودی n اگر 3 باشد 3 بار اجرا دارد پس داریم O(n) یعنی بسته به ورودی ما پیچیدگی زمانی تغییر می کند.

##### **How do the variables work in a Recursive Function?**

مثال مهم مقدار n در زمان return متد صدا زده شده به مقدار ما افزوده می شود

**using** *System;*

**namespace** *RecursionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(string[]** args**)**

**{**

**int** number = 5;

**int** Result = fun**(**number**)**;

Console.WriteLine**(**Result**)**;

Console.ReadKey**()**;

**}**

**static** **int** fun**(int** n**)**

**{**

**if(**n **>** 0**)**

**{**

**return** fun**(**n - 1**)** + n;

**}**

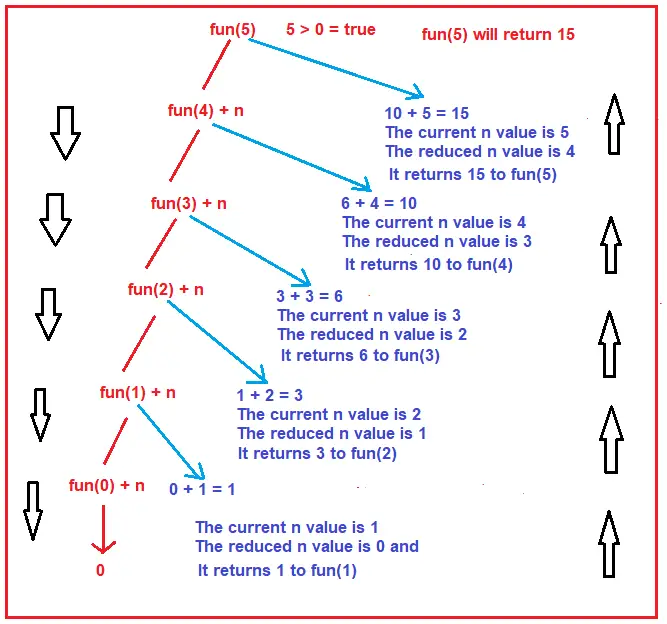
**return** 0;

**}**

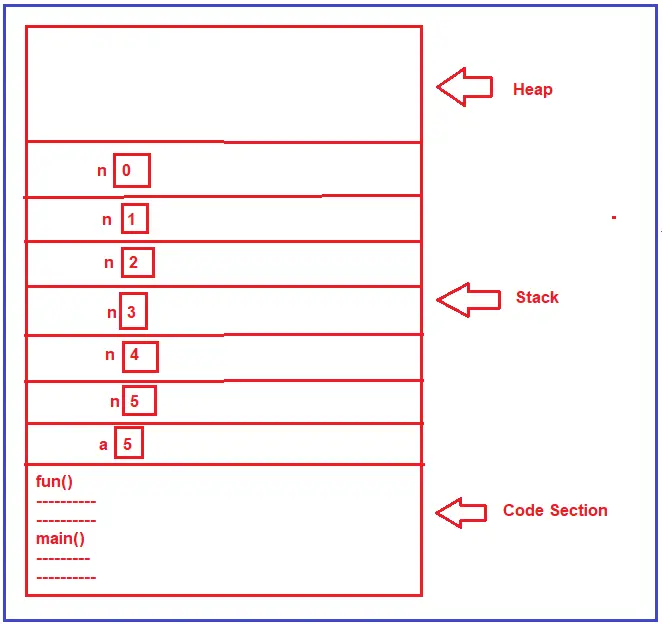
**}**

**}**

Trace در تصویر بعدی



دلیل stackOverflow شدن در حالت های recursive این است که متغییر در هر بار اجرا مقدار خود را در Stack نگه می دارد ولی در حلقه ها متغییر یک بار بیشتر در Stack مقدار دهی نمی شود.



نوع های داده در bcl در فضای نام system قرار دارند.

# **Command Line Arguments in C#**

راهکاری است برای ارسال parameter به متد main

نکته مهم این است که متد main از هیج متد دیگری paramer نمی گیرد. و فقط از command-line مقدار می گیرد.

از منوی properties و سپس تب Debug می توان command line arguemtns را مقدار دهی کرد که با استفاده از space مقادیر را از هم جدا می کند

# String in C#

کلاس است که دنباله ای از کاراکتر ها را ارائه می دهد و می توانیم عملیات های مختلفی بر روی آن انجام دهیم.

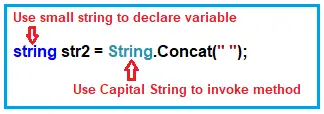
نکته » String از نوع reference type است

نکته » نوع های داده دیگر را اگر نگاه کنید می بینید که از نوع struct هستند به معنی value type اما string از نوع class است به معنی reference type بودن.

String vs string

می توان از هر دو استفاده کرد و تفاوتی ندارد در واقع string یک alias بر روی کلاس String است.

اما از String بهتر است برای method invoking استفاده کرد.



**Strings are Immutable in C#**

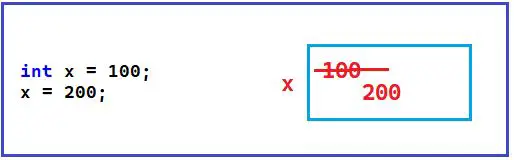
به معنی غیر قابل تغییر بودن string در سی شارپ است. به این معنی که وقتی رشته ای را به متغییری تخصیص می دهید که از نوع string است اگر مجدد مقدار جدیدی به آن تخصیص دهید مقدار قبلی pervious object به garbage collector رفته و یک object جدید ایجاد می شود و مقدار جدید را می گیرد. یعنی 2 مکان در حافظه ایجاد می شود. و هر بار که مقدار جدید به متغییر خود دهیم همین اتفاق رخ خواهد داد.



در واقع override اتقاق نمی اقتد برای مقدار قبلی بلکه حافظه جدید تخصیص داده می شود برعکس اتفاقی که در value type ها رخ می دهد. در این نوع override داده ی قبلی رخ می دهد.

Mutable به این معنی که داده ی آن با تغییر override می شود اما

Immutable داده ی آن با تغییر در حافظه ی جدید که ایجاد شده قرار می گیرد.



* مثال مدت زمان برای ایجاد حلقه ی بزرگ و تغییر آن در داده ی mutable , immutable
* **string** str = "";
* Console.WriteLine**(**"Loop Started"**)**;
* var stopwatch = new Stopwatch**()**;
* stopwatch.Start**()**;
* **for** **(int** i = 0; i **<** 30000000; i++**)**
* **{**
* str = Guid.NewGuid**()**.ToString**()**;
* **}**
* stopwatch.Stop**()**;
* Console.WriteLine**(**"Loop Ended"**)**;
* Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 25991

Now with integer

**int** ctr =0;

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000000; i++**)**

**{**

ctr = ctr + 1;

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 84

Not with same integer

**string** str = "";

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000000; i++**)**

**{**

str ="DotNet Tutorials";

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;

Time = 94

در این حالت به دلیل این که هر بار در حافظه مقدار جدید نمی سازد. به دلیل **String intern**

به این معنی که سی شارپ اگر مقدار رشته تغییر نکند از همان حافظه ی قبلی استفاده می کند. یعنی هر بار مقدار رشته را با مقدار رشته جاری بررسی می کند. برای افزایش performance

**StringBuilder for Concatenation in C#**

**نکته :** . The C# string immutability behavior can be very very dangerous when it comes to string concatenation

اگر یک رشته را مکررا با رشته ی دیگر ترکیب کنید هر بار مکانی جدید از حافظه اشغال می شود و مکان قبلی به g c تحویل داده می شود.

**string** str = "";

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000; i++**)**

**{**

str ="DotNet Tutorials" + str;

**}**

stopwatch.Stop**()**;

Console.WriteLine**(**"Loop Ended"**)**;

Console.WriteLine**(**"Loop Exceution Time in MS :" + stopwatch.ElapsedMilliseconds**)**;



String builder حافظه ی جدیدی نمی سازد و از همان مکان حافظه ی قبلی استفاده می کند و سبب افزایش شدید کارایی می شود. Time 1ms vs 5473

StringBuilder stringBuilder = new StringBuilder**()**;

Console.WriteLine**(**"Loop Started"**)**;

var stopwatch = new Stopwatch**()**;

stopwatch.Start**()**;

**for** **(int** i = 0; i **<** 30000; i++**)**

**{**

stringBuilder.Append**(**"DotNet Tutorials"**)**;

**}**

اما چرا string به صورت immutable است ؟ به دلیل thread safety به موقعیتی فکر کنید که چندی thread بخواهند یک string را تغییر دهند.

# Static Keyword in C#

اگر از هر developer بپرسید که چرا به static نیاز داریم می گویید که در factory و singleton به آن نیاز داریم و همچنین در data sharing اما 3 هدف مهم وجود دارد که در ادامه توضیح خواهیم داد.

نکته abstraction به این معنی است که هر چیزی که نیاز است از دید دیگران مخفی باشد را باید به این صورت پیاده سازی کرد.

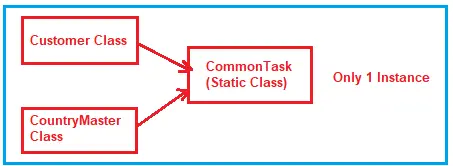
همچنین در oop باید در یک کلاس متد های که کاری مرتبط به هم انجام میدهند قرار داشته باشند مانند کی شی در دنیای واقعی.

نکته : زمانی که یک کلاس را به صورت static تعریف می کنیم یعنی کل اجزای کلاس نیز باید static باشد. در این حالت نیازی به new کردن کلاس برای دسترسی به اعضای آن نیست.

وقتی کلاس به static است clr یا common language runtime فقط یک نمونه از آن کلاس می سازد و به همه ارائه می دهد

نکته: ما هیج یک از قواعد oop را به کلاس static نمی توانیم پیاده سازی کنیم.

کلاس static یک بار instantiated می شود تا بتوانیم از اعضای آن استفاده کنیم.



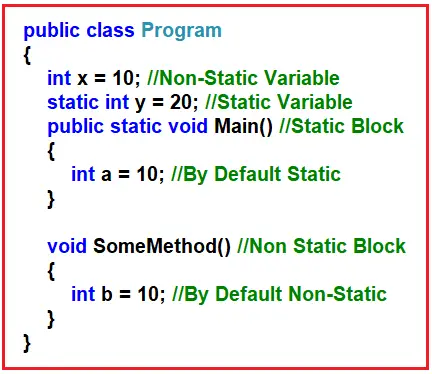
# Static vs Non-Static Members in C#

اعضای کلاس به 2 دسته بندی static و non static تقسیم می شوند.

Static : اعضای از کلاس که نیازی به ساخت instance. برای مقدار دهی یا اجرا ندارند را گویند.

اگر یک variable را به صورت staic تعریف کنیم یه در بلوک کدی که static است ایجاد کنیم نوع متغییر Static خواهد بود.

اعضای داخل متد static به صورت پیش فرض Static هستند.



مناسب برای نگه داری داده های از اپلیکیشن که در کل برنامه و برای همه ی object های یکسان است.

Static بلافاصله یک بار بعد از اجرای کلاس مقدار دهی میشود.

اما non static ها بعد از ایجاد object از کلاس مقدار دهی می شوند.

##### **What is the Scope of Static and Non-Static Variables in C#?**

Non staci با object ساخته می شود و با object ازبین می رود در واقع scope یک object برابر است با scope مربوط به reference variable

1. **Non-Static to Static:** Non-Static Members can be consumed only by using the object of that class inside a static block.
2. **Static to Static:** Static Members can be consumed directly (within the same class) or by using the class name (from outside the class) inside another static block.
3. **Static to Non-Static:** Static Members can be consumed directly or by using the class name inside a Non-Static block.
4. **Non-Static to Non-Static:** Non-Static Members can be consumed directly or by using the “this” keyword inside another non-static block.
5. **class** Example
6. **{**
7. **int** x = 100;
8. **static** **int** y = 200;
9. **static** **void** Add**()**
10. **{**
11. //we can access non static members X with the help of Example object
12. //We can access the static member directly or through class name
13. Example obj = new Example**()**;
14. //Console.WriteLine(obj.x + Example.y);
15. Console.WriteLine**(**"Sum of 100 and 200 is :" + **(**obj.x + y**))**;
16. Console.WriteLine**(**"Sum of 100 and 200 is :" + **(**obj.x + Example.y**))**;
17. **}**
18. **void** Mul**()**
19. **{**
20. //we can access static members directly or through class name
21. //we can access the non-static members directly or through this keyword
22. Console.WriteLine**(**"Multiplication of 100 and 200 is :" + **(**this.x \* Example.y**))**;
23. Console.WriteLine**(**"Multiplication of 100 and 200 is :" + **(**x \* y**))**;
24. **}**
25. **static** **void** Main**(string[]** args**)**

// Main method is a static method

1. // ADD() method is a static method
2. // Statid to Static
3. // we can call the add method directly or through class name
4. Example.Add**()**; //Calling Add Method using Class Name
5. Add**()**; //Calling Add Method Directly
6. // Mul() method is a Non-Static method
7. // We can call the non-static method using object only from a static method
8. // Static to Non-Static
9. Example obj = new Example**()**;
10. obj.Mul**()**; //Calling Mul Method using Instance
11. Console.ReadLine**()**;

##### **Example to Understand Static and Non-Static Constructors in C#**

نکته : سازنده غیر static اول از همه اجرا می شود.

**class** Example

**{**

//Static Constructor

//Executed only once

//First block of code to be executed inside a class

//Before Main Method body start executing, this constructor will execute

**static** Example**()**

**{**

Console.WriteLine**(**"Static Constructor is Called"**)**;

**}**

//Non-Static Constructor

//Executed once per object

//When we create an instance, this constructor will execute

**public** Example**()**

**{**

Console.WriteLine**(**"Non-Static Constructor is Called"**)**;

**}**

//Program Execution will start from the Main method

//But before executing the Main method body, it will

//execute the static constructor

**static** **void** Main**(string[]** args**)**

**{**

Console.WriteLine**(**"Main Method Execution Start"**)**;

Example obj1 = new Example**()**;

Example obj2 = new Example**()**;

Console.WriteLine**(**"Main Method Execution End"**)**;

Console.ReadLine**()**;

**}**

نکته » کلاس های static فقط اعضای static می توانند داشته باشند. و قابلیت ساخت instance ندارند

# Stack and Heap Memory in .NET

وقتی یک متغییر تعریف می کنیم چه اتفاقی رخ می دهد؟

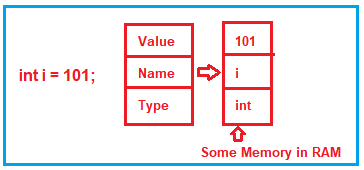
مقداری از حافظه ram را به متغییر تخصیص می دهد. این حافظه 3 چیز را بدنبال دارد

1 نام متغیرر

2 نوع داده متغییر

3 مقدار متغییر

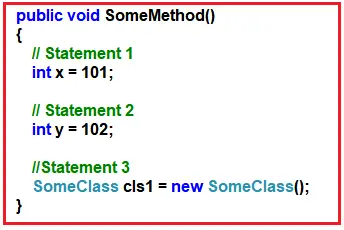
و وابسته به این که نوع داده value است یا reference حافظه stack یا heap اختصاص داده می شود.



##### **Understanding Stack and Heap Memory in C#**

2 نوع تخصیص حافظه در .net وجود دارد stack و heap

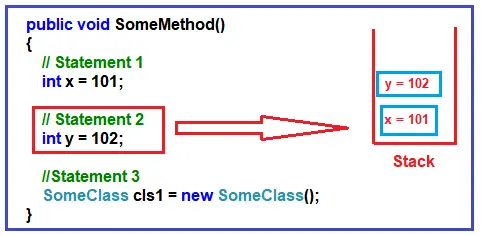
مثال



وقتی دستور اول اجرا می شود کامپایلر حافظه stack اختصاص می دهد.

وقتی دستور دوم اجرا می شود مجددا حافظه stack تخصیص داده می شود به این صورت که متغییر دوم در بالای متغییر اول در حافظه ی stack قرار می گیرد مانند ظرف های که روی هم قرار دارند.

تخصیص و عدم تخصیص حافظه stack به صورت last in – first out است یعنی از اول stack بر می دارد.



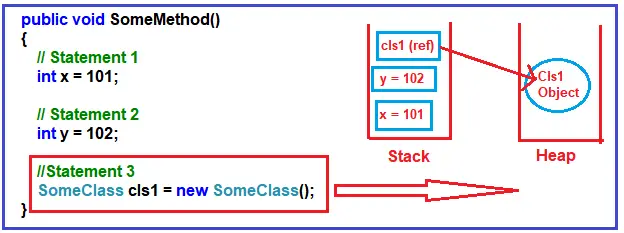
زمانی که دستور سوم اجرا می شود:

به صورت داخلی یک pointer در stack قرار می گیرد و object اصلی در مکان دیگری از حافظه ذخیره می شود به نام heap

Heap معمولا برای dynamic memory allocation کاربرد دارد.

نکته : reference pointer در Stack ذخیره می شود.

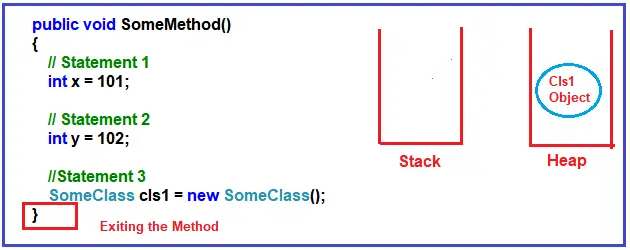
نکته : عبارت Person p1 هیچ حافظه ی را برای p1 اختصاص نمی دهد فقط یک variable با نام p1 در stack قرار می دهد و مقدار null را با آن تخصیص می دهد. و زمانی که کلمه کلیدی new برخورد می کند حافظه را در heap اختصاص می دهد.



##### **What Happens When the Method Completes Its Execution?**

وقتی اجرای متد به آخر می رسد و کنترل اجرا از متد خارج می شود یعنی به } می رسد. تمام variavble های ساخته شده در stack را پاک می کند به صورت lifo .

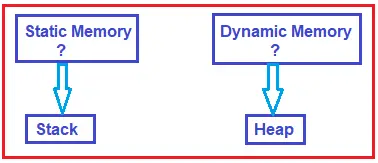
اما حافظه ی اختصاص داده شده به heap را از تخصیص خارج نمی کند. بعدا حافظه heap توسط garbage collector خالی می شود.



##### **Why do we have two types of memory?**

در سی شارپ primitive data type مثل int , double , bool تنها یک مقدار را نگه داری می کنند و در حالی که reference type ها یا object data type از نوع complex هستند یعنی یک نوع complex دارای reference به نوع های object دیگر و primitive های دیگر است.

بنابراین reference data type نگه دار ارجاع به چندین داده است و هر کدام از آن ها باید در حافظه ذخیره شود. نوع های object به dynamic memory نیاز دارند در حالی که primitive ها به static memory نیاز دارند.



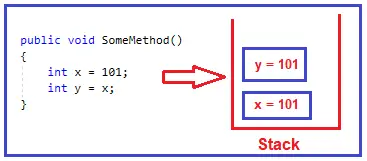
Value type vs reference type

Value ها نوع های هستند که هم Data و هم memory در یک مکان قرار دارند. اما Reference type نوع از داده هستند که دارای یک pointer که اشاره می کند به مکان اصلی حافظه.

##### **Understanding Value Type in C#:**

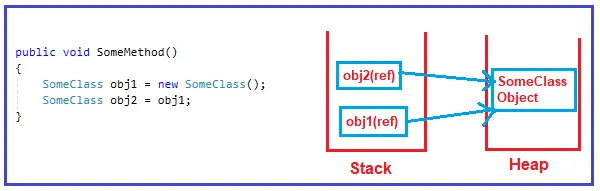
زمانی که به متغییری از نوع int مقداری را assign می کنیم و آن را به متغییر دیگری نیز تخصیص می دهیم یک copy از داده متغییر اول به متغییر دوم تخصیص داده می شود که کاملا از هم مستقل هستند. و تغییر یکی بر دیگری تاثیری ندارد مانند bool

Int double float و ...



##### **Understanding Reference Type in C#**

در این نوع اگر 2 متغییر از یک object ایجاد کنیم و سپس اولی را new کرده و مقدار آن را به دومی assign کنیم هر دو آن ها به یک مکان از حافظه اشاره خواهند کرد. و اگر یکی را تغییر دهید بر دیگری نیز تاثیر گذار خواهد بود مانند class interface object string , delegate



##### **How is the Heap Memory Freed Up?**

در حافظه ی Stack زمانی که کنترل اجرای برنامه در متد به اتمام می رسد و بعد از آخرین { شروع به deallocate کردن حافظه می کند اما در نوع heap این گونه نیست بلکه حذف حافظه با garbage است به این معنی که زمانی که object موجود در heap دیگر استفاده نمی شود به این معنی است که فاقد reference pointing شده و زمان حذف آن فرا رسیده.

Stack Key Points

* برای static memory allocation و local variable کاربرد دارد و توسط cpu مدیریت می شودکه سریع و کارآمد است.
* زمانی که یک متد فراخوانی می شود a block of memeory در stack برای local variable و parametrs اختصاص داده می شود و زمانی که به return می رسد بلوک غیر استفاده می شود و برای فراخوانی بعدی متد آماده می شود.
* متغییر های ذخیره شده در stack فقط در زمان فراخوانی متد در دسترس هستند.
* نگه داری value type or primitive و struct و reference object در آن انجام می شود

##### **Heap Memory Key Points:**

* برای dynamic memory که شامل object و complex data structures است و انعطاف پذیری بالا دارد و توسط garbage collector مدیریت می شود.
* Object ها در heap هستند و memory در runtime مدیریت می شود object جدید با کلمه کلیدی new ایجاد می شود و زمانی که استفاده نشوند gc آن را ازبین می برد.
* زمان آن ها از ایجاد با new تا زمانی که دیگر استفاده نشوند است.
* برای object array class , … استفاده می شود.

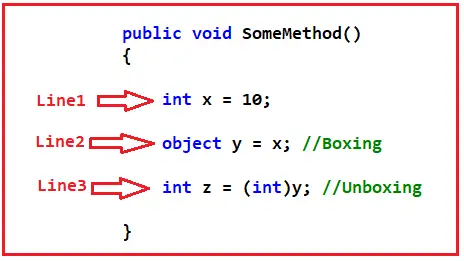
نکته : حافظه Stack فضای محدودی دارد ولی heap با توجه به نیاز رشد می کند. محدود آن حافظه کل است.

نکته : دسترسی به Stack سر راست تر است ولی دسترسی به heap دارای پیچیدگی بیشتر است.

# Boxing and Unboxing in C#

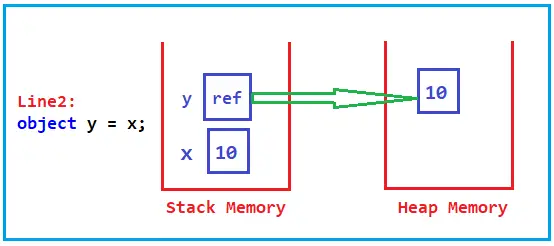
Boxing پروسه ی تبدیل value type به reference type است زمانی که Value type باکس می شود یک object جدید در heap تخصیص داده می شود و value در آن کپی می شود.

Unboxing برعکس boxing است زمانی که value از object استخراج می شود یعنی تبدیل reference به value و سبب کپی مقدار از heap به stack می شود

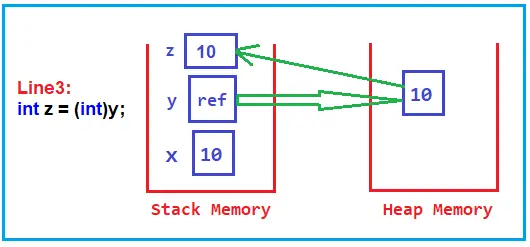


دستور اول متغییر x در حافظه stack ایجاد می شود و مقدار 10 را می گیرد

دستور دوم انتقال مقدار x که 10 است به نوع داده object انجام می شود و مقدار value type به reference type تخصیص داده می شود که عملیات boxing است



دستور سوم مقدار object که برابر 10 است را به مقدار z که از نوع value type است تخصیص می دهیم تبدیل reference به نوع value است عملیات unboxing



نکته عملیات boxing و unboxing که تبدیل مقدار value type به نوع reference type است می تواند روی عملکرد سیستم تاثیر گذار باشد و کاهش دهنده آن باشد.

به طور مثال 2 متد را 100000 بار اجرا کنید و در کی عملیات boxing را برای تخصیص مقدار یک متغییر به دیگری انجام دهید و در دیگری به طور ساده یک مقدار را از نوع value به Value دیگر تخصیص داده مدت زمان انجام عملیات متفاوت خواهد بود.

بهتر است انجام این نوع کار ها پرهیز شود. چون کاهش عملکرد دارد و تخصیص حافظه بالا چون در unbox حافظه heap و gc درگیر می شوند و همچنین unboxing نیازمند explicit case است که ممکن است exception دهد

نکته : در گذشته arrayList فقط object ذخیره می کردند که نیازمند عملیات boxing برای سایر نوع داده بوده اما اکنون generic وجود دارد از box پرهیز شده و نوع داده مستقیما در کالکشن ذخیره می شود.