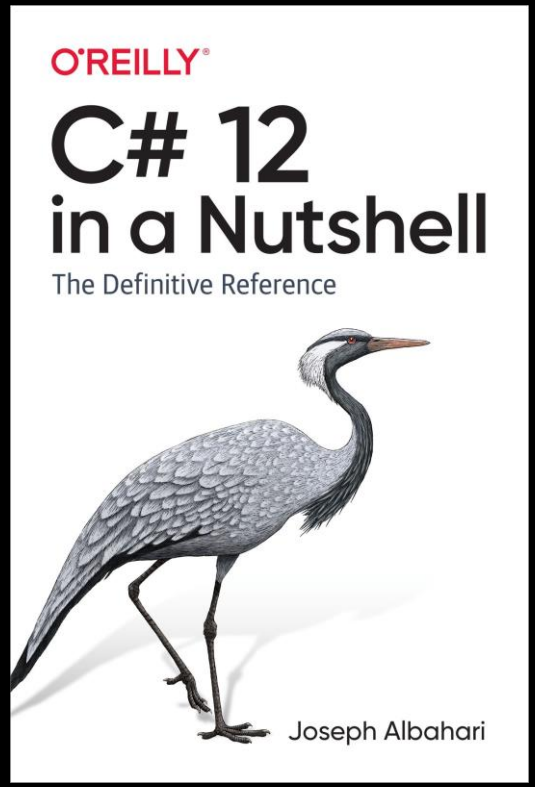
Variables and Parameters

- Stack
- Heap
- Garbage Collector (GC)



Variable یا متغیر چیست؟

A variable represents a storage location that has a modifiable value. A variable can be a local variable, parameter (value, ref, or out, or in), field (instance or static), or array element.

متغیر نماینده محل ذخیره شده یک مقدار است که مقدارش میتونه تغییر کنه. یک متغیر میتونه یه متغیر محلی باشه، میتونه یه پارامتر (منظورش ورودی متدهاست که میتونه معمولی یا out یا ref یا in) باشه، میتونه یه فیلد (استاتیک یا instance) باشه و یا میتونه یه خونه از یه آرایه باشه.

The Stack and the Heap

The stack and the heap are the places where variables reside. Each has very different lifetime semantics.

Stack و Heap محل هایی هستند که متغیرها در اونجا قرار داره. که از نظر lifetime خیلی باهم متفاوت هستند.

Stack

The stack is a block of memory for storing local variables and parameters.

Stack بخشی از حافظه برای نگهداری متغیرهای محلی و پارامترهاست.

The stack logically grows and shrinks as a method or function is entered and exited.

Stack خیلی منطقیه کلا، وقتی یک متغیر تو یه بلاک تعریف میشه رشد میکنه(حافظه اشغال میشه) و وقتی بلاک تموم میشه کوچک میشه(حافظه آزاد میشه)

Stack

```
1 reference | - changes | -authors, -changes
static int Factorial(int x)
{
    if (x == 0) return 1;
    return x * Factorial(x - 1);
}
```

برای مثال این کد رو ببینید:

این یک متد بازگشتی هستش که هربار خودشو صدا میزنه، (برای اینکه خیلی پیچیده نشه کاری با ورودی تابع نداریم)
با هر بار فراخوانی یک عدد (int) جدید در stack قرار میگیره و با هربار تموم شدن تابع فضا آزاد میشه.

(to avoid distraction, input argument checking is ignored)

This method is recursive, meaning that it calls itself. Each time the method is entered, a new int is allocated on the stack, and each time the method exits, the int is deallocated.

Heap

The heap is the memory in which objects (i.e., reference-type instances) reside.

Heap حافظه ایست که object ها مثل instance های جدید از یک کلاس در آن قرار میگیرند.

Whenever a new object is created, it is allocated on the heap, and a reference to that object is returned. During a program’s execution, the heap begins filling up as new objects are created.

به محض ایجاد یک شی، فضایی در heap به آن اختصاص داده میشه و یک reference که اونو برگردونه. در حین اجرای برنامه heap با شی های جدیدی که ساخته میشه شروع به پر شدن میکنه.

Garbage Collector (GC)

The runtime has a garbage collector that periodically deallocates objects from the heap, so your program does not run out of memory.

Runtime یه garbage collector داره(که بهش GC هم میگن) که به صورت دوره ای صدا زده میشه و object ها رو از رو heap پاک میکنه و فضا رو آزاد میکنه. اینجوریه که به خطای out of memory نمیخوره.

An object is eligible for deallocation as soon as it’s not referenced by anything that’s itself “alive.”

یک شی به محض اینکه دیگه هیچ رفرنسی بهش وجود نداشته باشه شرایط پاک شدن توسط GC رو پیدا میکنه

توضیح با مثال

```
StringBuilder ref1 = new StringBuilder("object1");
Console.WriteLine(ref1);
// The StringBuilder referenced by ref1 is now eligible for GC.

StringBuilder ref2 = new StringBuilder("object2");
StringBuilder ref3 = ref2;
// The StringBuilder referenced by ref2 is NOT yet eligible for GC.

Console.WriteLine(ref3);      // object2
```

بیاییم این تیکه کد رو باهم بررسی کنیم:

فرض کنید این کد همه چیزیست که ما توی program.cs نوشتیم. خط اول متغیر ref1 تعریف شده، خط دوم استفاده شده و دیگه تا آخر کد هیچ استفاده ای ازش نشده، پس بعد از خط دوم دیگه نیازی به این متغیر نداریم و GC میتونه فضای اشغال شده توسط این متغیر رو در heap آزاد کنه.

خط بعدی متغیر ref2 رو تعریف کردیم و خط بعد یه کپی ازش گرفتیم و ریختیم تو ref3، با وجود اینکه دیگه از ref2 استفاده ای نشده، این متغیر هنوز شرایط لازم برای آزادسازی heap رو نداره چرا که از ref3 هنوز داره استفاده میشه.