**Garbage collection (GITHUB) (test)**

Garbage collection nedir

In computer science, garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program.

Garbage collection is essentially the opposite of manual memory management, which requires the programmer to specify which objects to deallocate and return to the memory system. However, many systems use a combination of approaches, including other techniques such as stack allocation and region inference. Like other memory management techniques, garbage collection may take a significant proportion of total processing time in a program and, as a result, can have significant influence on performance.

-

-

-

**I read lot of articles about garbage collection and almost all article tells about heap memory. so my question is "garbage collection collects stack memory or heap memory or both"?**

**cvp:**

**It collects heap memory.**

**Usually, stack memory is collected *automatically* when the execution path reaches the end of the scope.** e.g.:

void fun()

{

int n; // reservation on the stack as part of the activation record

...

} **// returning the stack pointer to where it was before entering the scope**

In fact, in a language like C++, stack allocated variables are called auto variables.

**cvp:**

**Heap memory.**

Garbage collection is a method of deallocating memory that isn't being used anymore. Sometimes the "isn't being used anymore" part is tricky. **With the stack, as soon as a function returns, we can be confident (excepting programmer error) that the local variables aren't being used anymore, so they are deallocated automatically at that time in nearly every language/runtime.**

**cvp:**

**The stack is called a "stack" precisely because it is a zone of memory which is managed with a "stack policy", aka LIFO (as Last In, First Out). If allocation on the stack was not done in "the stack way" it would not be called a stack but heap.**

**Garbage Collection was invented in order to cope with the problem of allocating things on a heap, i.e. such that you cannot predict which parts will be released first. GC is meant for memory allocation problems where stack management is not sufficient.**

-

-

-

Automatic memory management is made possible by Garbage Collection in .NET Framework. When a class object is created at runtime, certain memory space is allocated to it in the **heap** memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

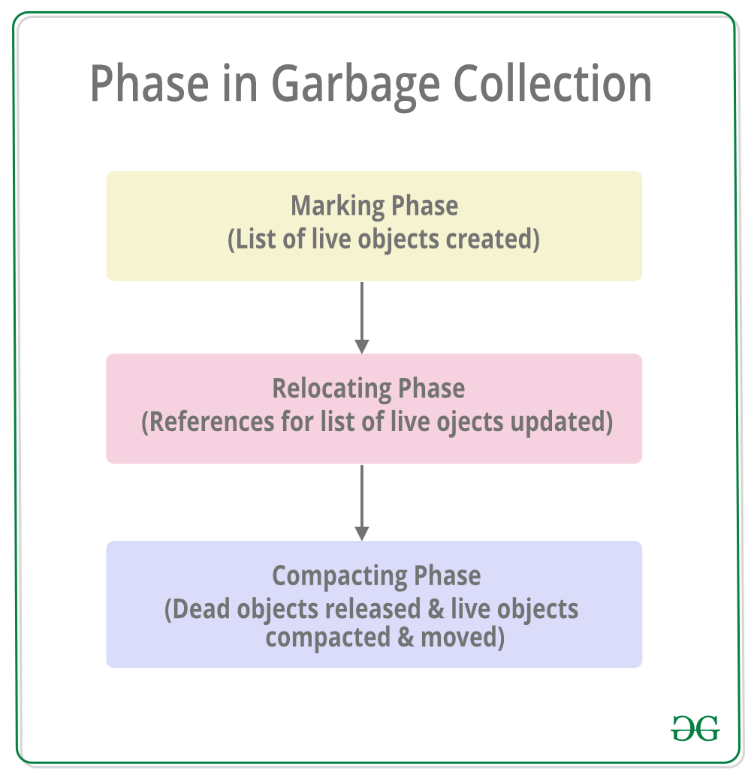
**Garbage collection will always work on Managed Heap** and internally it has an Engine which is known as the Optimization Engine.

Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

 If the system has low physical memory, then garbage collection is necessary.

 If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.

 If the ***GC.Collect*** method is called, then garbage collection occurs. **However**, this method is only called under unusual situations as normally garbage collector runs automatically.

****

**Marking Phase**: A list of all the **live** objects is created during the marking phase. This is done by following the references from all the root objects. **All of the objects that are not on the list of live objects are potentially deleted from the heap memory**.

**Relocating Phase**: The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.

**Compacting Phase**: The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are moved. All the live objects that remain after the garbage collection are **moved towards the older end of the heap memory in their original order.**

**Bu açıklama da iyi:**

C/C++/Pascal gibi dillerle yazılan uygulamalar direkt olarak makine koduna dönüştürülerek çalıştırılır. .NET, Java, Python gibi dillerde yazılan uygulamalar ise bir runtime tarafından çalıştırılırlar. runtime'a sahip olmanın en önemli avantajlarından biri uygulamanın hafıza yönetimini runtime'ın otomatik olarak yapmasıdır. runtime uygulamanın hafıza isteklerini işletim sisteminden önceden aldığı hafıza alanından karşılar, uygulamanın hafıza alanı ile işi bittiğinde ise geri alır. Hafızanın geri alınması görevini Garbage Collector - GC yapar.

GC otomatik hafıza yönetimi ile uygulama yazımını kolaylaştırmakta, kodlama süresini kısaltmaktadır. Yine GC, makine koduna derlenen dillere göre kodlama hatalarından dolayı karşılaşılan hafıza alanlarının leak edilmesi problemi ile de bir seviyeye kadar baş edebilmektedir.

GC çalıştığında uygulamada **artık referans edilmeyen** objeleri temizler, hafıza fragmantasyonunu engellemek için de temizlikle boşalan alanları hala kullanılmakta olan objeleri taşımak suretiyle birleştirir.

-

-

-

Even though with the .NET framework we don't have to actively worry about memory management and garbage collection (GC), we still have to keep memory management and GC in mind in order to optimize the performance of our applications. Also, having a basic understanding of how memory management works will help explain the behavior of the variables we work with in every program we write.  In this article I'll cover the basics of the Stack and Heap, types of variables and why some variables work as they do.

There are two places the .NET framework stores items in memory as your code executes.  If you haven't already met, let me introduce you to the Stack and the Heap.  Both the stack and heap help us run our code.  They reside in the operating memory on our machine and contain the pieces of information we need to make it all happen.

-

-

GARBAGE COLLECTOR NEDEN KULLANILMAMALI?:

It IS possible to use garbage collection with hard real time, if you have fully incremental garbage collector with bounded execution time per byte of allocated memory, so, crazily enough, it is NOT necessarily a reason not to use garbage collection :)

One fundamental problem with garbage collection, though, is that it is difficult to estimate and manage the actual size of the working set in memory, because garbage collector can free your memory only delayedly. So, yes, when memory is restricted, garbage collection might not be a good choice.

Another problem with garbage collection is that it sometimes interferes with freeing other resources such as file descriptors, window handles, etc., because, again, the garbage collector might free those resources only delayedly, causing resource starvation.

Garbage collection can also cause cache trashing, because the memory is not necessarily allocated in a local fashion. For example, stack allocated memory is much more cache-friendly than heap-allocated short-lived objects.

(bu kısmı saçma gibi. zaten adamın biri yorum yazmış buna😊 Finally, garbage collection of course consumes CPU time :) So if you can code manually memory management you can save the CPU cycles the garbage collector would consume :)

adamın yorum: Re: CPU time: manual memory management *also* takes up CPU time. Modern garbage collectors have an overhead of less than 3%. I doubt even the most experienced C programmer can get significantly below 1%. There's two ways you can look at this: either a garbage collector is twice as expensive as manual memory management, or the difference is negligible (~97% vs. ~99%).

-

GARBAGE COLLECTOR KULLANMAYAN YAZILIM DİLLERİ - NEDEN KULLANMIYOR (EN AŞAĞIDAKİ KIRMIZI TEXTE BAK)

Languages like C and C++ use dynamic heap allocation through dedicated functions/operators like malloc and new. This allocates memory on the heap, in RAM. If such a program fails to free the memory once done using it, then the programmer has managed to create a certain kind of bug called memory leak. Meaning that the program now consumes heap memory that cannot be used, since there is nothing in the program pointing at it any longer.

However, all memory allocated by a process is freed by the OS when the process is done executing. If the process failed to clean up its own heap allocations, the OS will do it. It is still good practice to manually clean up the memory though, but for other reasons (exposes latent bugs).

Therefore the only concern with memory leaks is that they cause programs to consume too much RAM while they execute. Once the process is done executing, all memory - including leaked memory - is freed.

There is no relation between the heap and your hard drive, just as there is no relation between the stack and your hard drive. The hard drive is used for storing the executable part of your program, nothing else. The heap, stack and other such memory areas are for storing data when your program is executing. Since they are allocated in RAM, all info in such areas is lost when the program is done executing.

The reason why some languages introduced garbage collection, was to remove the problem with memory leaks. A garbage collector is a background process of sorts, that goes through a program's heap memory and looks for segments of data which no part of the program is pointing at, then free those segments. Since the garbage collector does this, there is no need for free()/ delete.

This comes at the expensive of execution speed, since the garbage collector needs to be executed now and then. This is one of many reasons why languages like Java and C# are slower than C and C++ by design. And it is also the reason why C and C++ don't have and never will have a garbage collector, since those languages prioritize execution speed.

–

If you don't free a resource, it stays allocated. The C compiler knows nothing about hard drives. You can read and write files with C and the appropriate IO libraries. So yes, your hard disk might be littered with stuff from running your software, but the C language or compiler isn't responsible to clean it up. You are. You can clean up your files manually, or code your C programs to clean up after themselves. Get a good book on C.

-