What's the use of C# keyword fixed/unsafe? [duplicate]

Asked 10 years, 10 months ago Modified 1 year, 9 months ago Viewed 37k times

32

This question already has answers here:

C# Unsafe/Fixed Code (7 answers)

Closed 10 years ago.

What's the use of C# keyword fixed/unsafe?

For example, *C# fixed Keyword (unsafe)*

```
using System;

class Program
{
    unsafe static void Main()
    {
        fixed (char* value = "sam")
        {
            char* ptr = value;
            while (*ptr != '\0')
            {
                Console.WriteLine(*ptr);
                ++ptr;
            }
        }
    }
}
```

Why do I need to fix it in the first place?

c#

Share

Improve this question

Follow

edited May 30, 2022 at 19:51

Peter Mortensen

31.6k • 22 • 109 • 133

asked Feb 24, 2014 at 7:12

Swab.Jat

1,270 • 1 • 14 • 20

see: stackoverflow.com/questions/153376/fixed-statement-in-c-sharp or stackoverflow.com/questions/85479/c-sharp-unsafe-fixed-code or stackoverflow.com/questions/3173002/unsafe-code-in-c-sharp – Belial09 Feb 24, 2014 at 7:14

6 You need to fix it so that the pointer you're using will stay valid. But the uses of unsafe code are relatively few and far between, to be honest. I've been writing C# for many years and still don't even remember the various rules for unsafe coding. – Jon Skeet Feb 24, 2014 at 7:14

check this: msdn.microsoft.com/en-us/library/f58wzh21.aspx – w.b Feb 24, 2014 at 7:14

@TimurAykutYıldırım I do not think that is a duplicate - that question asks when this asks why as well. – markmnl Feb 24, 2014 at 7:31

1 No answer explains this statement found in the linked example: "*There is a small cost to using the fixed statement. So **it will only help** on operations that spend significant amounts of time in unsafe code*" which is at least ambiguous, maybe wrong, depending on how you read the first part. What is unclear is whether "non fixed" pointers are adjusted by the CLR when GC relocates the allocated memory block (at the cost of some time loss), or are not and start pointing to a wrong place (hence the 'fixed' keyword is not *optional* but mandatory with pointers -- this is my understanding). – mins Apr 12, 2019 at 8:32

3 Answers

Sorted by: Highest score (default)

58

C# is a *managed language* that means the memory is managed automatically, i.e. not by you. If you did not use `fixed` by the time you come to modify the memory pointed to by your pointer C# could have moved the variable to another memory location so you could be modifying something else!

`fixed` is logically fixing the variable in memory so it does not move around.

Why does C# move variables in memory around? To compact the memory otherwise programs would use up more memory available to them if objects that are no longer alive left holes other objects cannot fit in (heap memory fragmentation).

I used `fixed` extensively in a .NET library designed for resource constrained devices to avoid creating garbage copying into buffers and find this feature sorely lacking in other managed languages where you cannot do the same. When writing games in a managed language garbage collection is often one of the biggest bottlenecks so having the ability not to create it is very helpful!

See my question here: C# Copy variables into buffer without creating garbage? for one reason why.

Share

Improve this answer

Follow

edited Mar 26, 2023 at 0:39

answered Feb 24, 2014 at 7:22

markmnl

11.4k ● 10 ● 76 ● 113

5 "I used fixed extensively in a .NET library designed for resource constrained devices to avoid creating garbage copying into buffers and find this feature sorely lacking in other managed languages where you cannot do the same. When writing games in a managed language garbage collection is often one of the biggest bottlenecks so having the ability not to create it

is very helpful!" --> Thank you, but sorry can you elaborate. I seems to get the idea, really pinning memory location is for Performance. But I don't really... see how, would appreciate if you elaborate on this! – Swab.Jat Feb 24, 2014 at 7:57

2 Pinning memory doesn't give the performance the ability to use pointers to modify memory, in my case copy several variables into a buffer without creating garbage can give performance where garbage collection time is an issue, see my linked question. – markmnl Feb 24, 2014 at 8:03

Thanks Mark. Read it. How much performance gain you had I wonder by not using intermediate byte[] and copy via pined memory location instead? Was it significant? – Swab.Jat Feb 24, 2014 at 8:22

If you are on Phone or device like Ouya (running mono) and copying a lot it was noticeable - the app freezes when the GC has to run - now it doesn't because there is no garbage. – markmnl Feb 24, 2014 at 8:28

Great appreciate your input. – Swab.Jat Feb 24, 2014 at 8:43

14

`unsafe` is necessary to deal in pointers.

`fixed` has two uses:

- it allows you to pin an array and obtain a pointer to the data
- when used in an `unsafe struct` field, it declares a "fixed buffer" - a reserved block of space in a type that is accessed via pointers rather than regular fields

To answer with a specific example - here's some code that is used to perform semantic equality between two `byte[]` of arbitrary size...

```
internal static unsafe  int GetHashCode(byte[] value)
{
    unchecked
    {
        if (value == null) return -1;
        int len = value.Length;
        if (len == 0) return 0;
        int octects = len / 8, spare = len % 8;
        int acc = 728271210;
        fixed (byte* ptr8 = value)
        {
            long* ptr64 = (long*)ptr8;
            for (int i = 0; i < octects; i++)
            {
                long val = ptr64[i];
                int valHash = (((int)val) ^ ((int)(val >> 32)));
                acc = (((acc << 5) + acc) ^ valHash);
            }
            int offset = len - spare;
            while(spare-- != 0)
            {
                acc = (((acc << 5) + acc) ^ ptr8[offset++]);
            }
```

```
            }
            return acc;
        }
    }
```

So if, for example, the buffer was 1000 items, by treating it as a set of `long` we now only do 125 iterations rather than having to look individually at all 1000 - plus we completely bypass any array bounds checking (which the JIT may or may not remove, depending on how *obvious* it looks that you can't possibly be violating them).

Share Improve this answer Follow

answered Feb 24, 2014 at 7:35

Marc Gravell
1.1m ● 273 ● 2.6k ● 3k

Thanks Marc - "for example, the buffer was 1000 items, by treating it as a set of long we now only do 125 iterations rather than having to look individually at all 1000" --- not sure if I follow you but I am guessing you're saying it's basically "Performance Consideration". If you have a big object graph - sometimes, it's faster if you pin/fix its memory location while iterating over it? If so it's the only "Practical" usage I hear thus far (Everyone else just reiterate "yes fix it in memory") – Swab.Jat Feb 24, 2014 at 7:54

2 @Swab.Jat if you look at the code, you can see that it is processing each item as a `long` rather than as a `byte` - so it is hashing 8 bytes each iteration. Yes, `unsafe` is usually used as a performance consideration. There is also the scenario where you obtain a pointer to completely unmanaged memory - interop scenarios, basically - but those are much rarer than performance uses. – Marc Gravell Feb 24, 2014 at 10:06

hi Marc, thanks. Mark gave me a very useful example - stackoverflow.com/questions/15307431/... – Swab.Jat Feb 24, 2014 at 10:38

4

`Unsafe` code blocks are quite rare to be used. They are predominantly used if you want to create and make use of pointers. Also, any code in unsafe block is out of control of CLR. It is considered to be unverifiable (from MSDN) as far as CLR in concerned.

Now, when garbage collection happens, some object may well be relocated. When prefixed with fixed, we are telling the framework to not to relocate objects whose address pointers are pointing to.

Share Improve this answer Follow

answered Feb 24, 2014 at 7:21

danish
5,600 ● 2 ● 26 ● 28

1 Note that "Fixed" ia an interop issue. WHen you ge a pointer and hand it over to unmanaged code (for example to a native dll to dump some data there) you CAN NOT have the garbage collector move that object while your call is in progress. Hence fixing it on a location.
– TomTom Feb 24, 2014 at 7:27

1 That's what MSDN says - fix memory location yes I get it. But why? Any practical usage why we need this language feature at all? Thanks – Swab.Jat Feb 24, 2014 at 7:55

1 Mostly while doing interop and marshalling. When you want to retain pointers for longer durations. – danish Feb 24, 2014 at 8:47