Using Pointers in C#

Phil Curnow Updated date Feb 08, 2022

Back in the late 1980's and up to around 1997, most of my development work was done using the C programming language. Anyone who has done a fair amount of work in C or C++ would not have been able to avoid the use of pointers. Recently I had to re-visit some old code I had written in C and one of my first comments was 'wow pointers, I remember them!'

I have known since I started using C# (and that's been a few years) that you could use pointers, but to be honest, I have never had to use them and have never gone looking for a reason to use them. The main reasons I have written this article is to introduce the concept of pointers to any developers who have not seen them before, and as a little nostalgic trip back to the days when they were part of my daily coding diet!

So, what is a Pointer?

A pointer is nothing more than a variable that holds the address in memory of another variable. In C#, pointers can only be used on value types and arrays. As a structure is a value type, pointers can be used with them, but there is one caveat with this, the structure must not contain any reference types if you plan to use pointers. Any of the following may be a pointer:

- Sbyte
- byte
- short
- ushort
- int

- uint
- long
- ulong
- char
- float
- double
- decimal
- bool

unsafe keyword

Typically when we write code in C# by default it is what's known as safe code. Unsafe code allows you to manipulate memory directly, in the normal world of C# and the .NET Framework, this is seen as potentially dangerous, and as such you have to mark your code as unsafe. Using unsafe code, you are loosing garbage collection and whilst directly accessing memory, you have the possibility of accessing memory that you didn't want to and causing untold problems with your application. Unsafe code can only be used within a fully trusted assembly.

Now it is time to write some program code to demonstrate the use of pointers. Consider the following code.

```
1  static void Main(string[] args)
2  {
3      int age = 32;
4      Console.WriteLine("age = {0}", age);
5  }
```

A very simple piece of code that will print **age = 32** on the console. Now let's introduce a pointer into the mix. When we declare a pointer, we have to declare it in a certain way, this being type* variable; the asterisk (dereferencer symbol) informs the compiler that we are declaring a pointer variable, the type says that we intend to use a pointer to store the address of type type. So using our simple piece of code above, let's creates an integer pointer to point to age.

```
1  static void Main(string[] args)
2  {
3      unsafe
4      {
5          int age = 32;
6          int* age_ptr;
7          Console.WriteLine("age = {0}", age);
8      }
9  }
```

Notice the use of the unsafe keyword, if we had not enclosed our code within unsafe, we would have received the following compilation error.

Pointers and fixed sized buffers may only be used in an unsafe context.

We could have declared the whole method as unsafe and this would have worked fine, for example:

```
1  unsafe static void Main(string[] args)
```

So, what can we do with our pointer? As stated at the beginning of the article, pointers hold addresses in memory to other variables, so we need to make age_ptr point to the memory location of age. To do this, we use the following line of code:

```
1  age_ptr = &age;
```

The ampersand '&' is the referencer and means the 'location of'. So if we now do the following in our code:

```
1  static void Main(string[] args)
2  {
3      unsafe
4      {
5          int age = 32;
6          int* age_ptr;
7          age_ptr = &age
8          Console.WriteLine("age = {0}", age);
9          Console.WriteLine("age_ptr = {0}", *age_ptr);
10     }
11 }
```

We should now see the following output on the console:

age = 32
age_ptr = 32

Basically, with the display line for age_ptr, we are printing the value held in the memory location that age_ptr is pointing to (in this instance, the age variable).

Now, what happens if we add the following to the example?

```
1  *age_ptr += 3;
2  Console.WriteLine("age now = {0}", age);
```

What do you think the output will be (*note that we are printing the value of age*)? If you have said 35 you are correct! The value of age has been changed to 35. Let's dissect the line *age_ptr += 3

and see what's happening.

age_ptr has been declared as an integer, the term *age_ptr can be read as 'the integer value at the memory location age_ptr is pointing to'. We are then simply adding 3 to the integer value, giving us a result of 35. As age_ptr is pointing to the memory location where age stores its value, essentially we have modified the variable age.

Pointers to Structures

As previously mentioned, you can define a pointer to a structure, but you have to make sure that there are no reference types in the structure, for example, consider the example code below:

```
1  class TestClass
2  {
3      . . .
4  }
5
6  struct TestStruct
7  {
8      public int age;
9      public TestClass test;
10 }
```

You would not be able to then do the following:

```
1  TStruct test_struct;
2  TStruct* struct_ptr;
3  struct_ptr = &test_struct;
```

When compiling this code, you would receive the error message:

Cannot take the address of, get the size of, or declare a pointer to a managed type.

So let's create a valid structure and see how we use a pointer to the structure. Our structure will be a simple one that contains two integers X and Y to simulate co-ordinates on a screen.

```
1  struct Location
2  {
3      public int X;
4      public int Y;
5  }
```

Now we can create the following code:

```
unsafe static void Main(string[] args)
{
    Location loc;
    loc.X = 100;
    loc.Y = 100;

    Location* loc_ptr;
    loc_ptr = &loc;
}
```

There should be no surprises in the code above. loc_ptr is of the type Location, which is our structure. So how do we use our pointer? To access the X and Y members of our structure through the use of our pointer, we can use the following syntax

```
(*loc_ptr).X;  or  loc_ptr->X;
```

I know which one I would rather use (the second option), but both perform exactly the same function. So if we wanted to display and modify X we can do this as follows:

```
unsafe static void Main(string[] args)
{
    Location loc;
    loc.X = 100;
    loc.Y = 100;
    Location* loc_ptr;
    loc_ptr = &loc;
    Console.WriteLine("X = {0}", loc_ptr->X);
    loc_ptr->X = 200;
    Console.WriteLine("X = {0}", loc_ptr->X);
}
```

Pointers and Classes

A class is a reference type, so therefore you cannot define a pointer to a class, but you can define a pointer to a class member that is a value type. Before we see an example of how to do this, there is something that you need to understand before creating pointers to value type members of a class.

When using safe code, the garbage collector will possibly move an object in memory during its lifetime, this is done to manage free resources and stop fragmentation. If you have a pointer to a value type in a class, this could cause some unexpected results, and some headaches! Needless to say, there is a way around this problem. We can instruct the garbage collector not to move specific objects. We do this by using the fixed keyword. Let's take a look at an example.

The design of the following class is not ideal and any OO design guru's out there will probably be shaking their heads at it, but it has been done as a very simple example.

For Example

```
class PTest
{
    public PTest()
    {
    }
    public int age;
    public void DisplayAge()
    {
        Console.WriteLine("age = {0}", age);
    }
}
```

And some example code to create an instance of the class and then manipulate the public variable age using a pointer.

```
unsafe static void Main(string[] args)
{
    PTest p = new PTest();

    p.age = 32;
    p.DisplayAge();

    fixed (int* age = &p.age)
    {
        *age += 3;
    }

    p.DisplayAge();
}
```

Notice the use of the keyword fixed. Basically what this is ensuring is that during the execution of the code within the fixed statement, the object p will not be moved, thus solving the problem of our pointer pointing to an invalid memory location, this operation is called 'pinning'.

Compilation of unsafe code

In order to compile program code that contains unsafe code, you need to inform the compiler. When using the command line compiler, you need to issue the /unsafe switch. When using Visual

Studio, you need to tick the box in the project properties titled Allow Unsafe code (this is found within the Build section).

Conclusion

Using pointers in C# is almost never required and they are generally only used in conjunction with P/Invoke. If you have a requirement to use pointers, you need to be very careful as you are manipulating memory directly.

My advice would be, don't go looking for a reason to use them, if you can perform your task using safe code, do so.

.Net C# c# pointers pointers

Next Recommended Reading
Pointers In C#

Phil Curnow
I work as a consultant in the UK with a large software and services company. I have been in the IT industry as a consultant and developer for over 20 years. I have worked with many programming languages over the years bu... Read more
http://www.curnow.biz

Type your comment here and press Enter Key (Minimum 10 characters)

Thanks for the beautifully explained article. I want to know that when we use fixed keyword then our current instance will not be moved to garbage. But we are displaying the contents of age variable by using p.DisplayAge(). How is this possible? Suppose the p object is moved to garbage after the code snippet of fixed region. Then how can we access the age variale?

Neil Johnson
2003 124 355

May 07, 2018
0 0 Reply

The article is very helpful but not in the context of the use in Winforms. I need to do something in a class that is not static as is the case in Main.

I need to take a pointer that is returned from a legacy app that uses realloc. I can use P/Invoke to accomodate the mem function, but I am having problems with the pointer fro the function.

The function call is:
MemBlockList = (pMemBlock)realloc(MemBlockList, (MemBlockCount+1) * sizeof(sMemBlock));

Where:
sMemBlock and pointer to it is a struct of:

```
struct     sMemBlock {
   WORD    Base;
   BYTE    Length;
   BYTE    Original[16];
   BYTE    Support[16];
   };
typedef    sMemBlock*     pMemBlock;

unsigned   MemBlockCount  = 0;
pMemBlock  MemBlockList   = NULL;
```

How do I do this?

Al S
2124 3 0

Jul 08, 2009
0 0 Reply

I have an api dll written in c++. One of the functions is imported in c++ like this:

typedef Parameter*(_stdcall * ExtractParameterType)(void) ;

Where Parameter is:

```
typedef struct tagParameter
{
char Name[18] ;
char Val[18];
char Unit[16];
char High[18];
char Low[18];
```

char Flag;
} Parameter;

In C# i wrote it this way:

[DllImport("MidCommLib.dll")]
public static extern Parameter[] ExtractParameter();

Where Parameter is:
public struct Parameter
{
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 18)]
public string Name;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 18)]
public string Val;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 16)]
public string Unit;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 18)]
public string High;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 18)]
public string Low;

[MarshalAs(UnmanagedType.U1)]
public char Flag;
};

In the main code I use this:
Parameter[] para = BcComm.ExtractParameter();

This doesn't seem to work :(

How can I a pointer in C#, as a structure array, so that I can retrieve the results returned by ExtractParameter() ?

Thank you,
Augustin

Augustin Jianu
2126 1 0

Jan 23, 2008
0 0 Reply

"Cannot take the address of, get the size of, or declare a pointer to a managed type ('WEP.frmWep.wep_key')" Cause of this error?

nachi
2116 11 0

Dec 08, 2007
0 0 Reply

If you could supply more code, perhaps I can help you out.

Phil Curnow

Dec 10, 2007

FEATURED ARTICLES

Create An Android App With .NET MAUI And Visual Studio 2022

Sealed Classes - A Java 17 New Feature

CRUD Operation And Microservice Communication Using gRPC In .NET Core 6 Web API

How To Post Data To The Controller Using AJAX With Validations In ASP.NET Core

JWT Token Creation, Authentication And Authorization In ASP.NET Core 6.0 With Postman

TRENDING UP

01 Build A Blazor Hybrid App with .NET MAUI for Cross-Platform Application

02 Python Constants

03 Getting Started With .NET MAUI For Cross Platform Application Development

04 Asynchronous Communication Between Microservices Using .NET Core API, RabbitMQ, and Docker

05 What Is DMARC ? | Why DMARC Is Important ? | Understanding DMARC Records

06 SQL - Clone Tables😉

07 Create QR Code Using Google Charts API In VB.Net

08 gRPC Introduction And Implementation Using .NET Core 6

09 Implementation And Containerization Of Microservices Using .NET Core 6 And Docker

10 Index() Method In Python

Learn C# 8.0

CHALLENGE YOURSELF

C# Skill

GET CERTIFIED

JavaScript Developer