



Krishna Nigalye

Software Developer

# Learning C-Sharp

## Method Argument Passing

### PASSING ARGUMENTS TO METHOD PARAMETERS

- In C#, arguments can be passed to a method either by value or by reference.

Let's see the difference between *pass by value* and *pass by reference*.

- By default, arguments are passed by value. In this a copy of data is created and passed to the method. So, any changes that can happen to those arguments inside the function will not be reflected back to the original value of arguments.

Check out this example for pass by value.

```
Program.cs
1 using System;
2
3 namespace c_sharp_out_ref_keywords
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             TestValueType.TestPassByValue();
10        }
11    }
12 }
```

```
TestValueType.cs
1 using System;
2
3 namespace c_sharp_out_ref_keywords
4 {
5     public class TestValueType
6     {
7         private static void FindSquare(int num)
8         {
9             num *= num;
10            Console.WriteLine($"=> Value of num inside FindSquare() is {num}");
11        }
12
13        public static void TestPassByValue()
14        {
15            Console.WriteLine("Pass by Value");
16            Console.WriteLine("-----");
17
18            int num = 5;
19            Console.WriteLine($"=> Value of num before calling FindSquare() is {num}");
20
21            FindSquare(5);
22
23            Console.WriteLine($"=> Value of num after calling FindSquare() is {num}");
24
25            Console.ReadLine();
26        }
27    }
28 }
```

```
Pass by Value
-----
=> Value of num before calling FindSquare() is 5
=> Value of num inside FindSquare() is 25
=> Value of num after calling FindSquare() is 5
```

There are two ways to pass value by reference using ref and out keywords. But there is a difference between them. Let's see it with the help of an example.

- Ref keyword:
  - You have to use ref keyword in the method definition as parameters.
  - The argument you are passing as a ref parameter has to be initialized or has to have a value before the function call.
  - Compiler does not generate any error if you fail to alter value of the parameter marked as ref.

The screenshot displays two C# source files in Visual Studio. The left file, `Program.cs`, defines a `namespace c_sharp_out_ref_keyworc` with a `class Program` containing a `static void Main(string[] args)` method that calls `TestRefKeyword.TestPassByReference();`. The right file, `TestRefKeyword.cs`, defines a `namespace c_sharp_out_ref_keywords` with a `public class TestRefKeyword`. Inside this class, there is a `private static void FindSquare(ref int num)` method that squares the value of `num` and prints it. Below this is a `public static void TestPassByReference()` method that prints "Use of Ref keyword", a separator line, the value of `num` before calling `FindSquare`, calls `FindSquare(ref num)`, prints the value of `num` after the call, and reads a line from the console. A console window at the bottom shows the output of the program.

```
Program.cs
1 namespace c_sharp_out_ref_keyworc
2 {
3     class Program
4     {
5         static void Main(string[] args)
6         {
7             TestRefKeyword.TestPassByReference();
8         }
9     }
10 }
11
```

```
TestRefKeyword.cs
1 using System;
2
3 namespace c_sharp_out_ref_keywords
4 {
5     public class TestRefKeyword
6     {
7         private static void FindSquare(ref int num)
8         {
9             num *= num;
10            Console.WriteLine($"=> Value of num inside FindSquare() is {num}");
11        }
12
13        public static void TestPassByReference()
14        {
15            Console.WriteLine("Use of Ref keyword");
16            Console.WriteLine("-----");
17
18            int num = 5;
19            Console.WriteLine($"=> Value of num before calling FindSquare() is {num}");
20
21            FindSquare(ref num);
22
23            Console.WriteLine($"=> Value of num after calling FindSquare() is {num}");
24
25            Console.ReadLine();
26        }
27    }
28 }
29
```

```
Use of Ref keyword
-----
=> Value of num before calling FindSquare() is 5
=> Value of num inside FindSquare() is 25
=> Value of num after calling FindSquare() is 25
```

- Out keyword:
  - You have to use out keyword in the method definition as parameter.
  - Unlike ref, the variable to be passed need not be initialized not be declared before the function call.
  - With that said, the parameter declared as an out parameter has to have a value before returning.

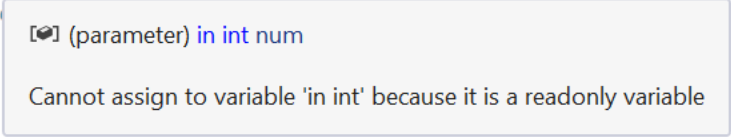
```
Program.cs
1
2 namespace c_sharp_out_ref_keywords
3 {
4     class Program
5     {
6         static void Main(string[] args)
7         {
8             TestOutKeyword.TestPassByReference();
9         }
10    }
11 }
12
```

```
TestOutKeyword.cs
1 using System;
2
3 namespace c_sharp_out_ref_keywords
4 {
5     public class TestOutKeyword
6     {
7         private static void FindSquare(out int num)
8         {
9             num = 5; // value is initialized inside the function
10            num *= num;
11            Console.WriteLine($"=> Value of num inside FindSquare() is {num}");
12        }
13
14        public static void TestPassByReference()
15        {
16            Console.WriteLine("Use of Out keyword");
17            Console.WriteLine("-----");
18
19            FindSquare(out int num);
20
21            Console.WriteLine($"=> Value of num after calling FindSquare() is {num}");
22
23            Console.ReadLine();
24        }
25    }
26 }
```

```
Use of Out keyword
-----
=> Value of num inside FindSquare() is 25
=> Value of num after calling FindSquare() is 25
```

- In keyword:
  - This is a new keyword which is introduced in C# 7.2.
  - It allows you pass an argument to a method by reference but only as **readonly**.
  - Any attempt to change the value which is pass by reference as 'in' parameter will result in compile error.

```
private static void PrintNumbers(in int num)
{
    num = num + 5;
    Console.WriteLine("Numbers() is {num}");
}
```



You may think that why do we need a pass by reference but as a 'readonly' and if its readonly why not to pass it by a value. Right?

Well you can think of it this way. You can imagine a situation where you are passing a big value type object to a function which is called multiple times. So, using 'in' keyword will allow you to pass this object by reference instead of creating a new copy of that object in each call.

Look at the following code which makes use of 'in' parameter.

TestInKeyword.cs

c-sharp-out-ref-keywords

7

namespace c\_sharp\_out\_ref\_keywords

8

{

9

public class TestInKeyword

10

{

11

private static void PrintNumbers(in int num)

12

{

13

Console.WriteLine(\$"=> Value of num inside PrintNumbers() is {num}");

14

}

15

public static void TestPassByReference()

16

{

17

Console.WriteLine("Use of In keyword");

18

Console.WriteLine("-----");

19

for (int num = 0; num <= 100; num++)

20

{

21

Console.WriteLine(\$"=> Value of num before calling FindSquare() is {num}");

22

PrintNumbers(in num);

23

Console.WriteLine(\$"=> Value of num after calling FindSquare() is {num}\n");

24

}

25

}

26

}

27

}

28

}

29

}

30

}

31

}

32

}

108 %

No issues found

Program.cs

c-sharp-out-ref-keywords

3

namespace c\_sharp\_out\_ref\_keywords

4

{

5

class Program

6

{

7

static void Main(string[] args)

8

{

9

TestInKeyword.TestPassByReference();

10

Console.ReadLine();

11

}

12

}

13

}

14

}

15

}

C:\Users\kpnigalye\Documents\\_programming\c...

=> Value of num inside PrintNumbers() is 55

=> Value of num after calling FindSquare() is 55

=> Value of num before calling FindSquare() is 56

=> Value of num inside PrintNumbers() is 56

=> Value of num after calling FindSquare() is 56

=> Value of num before calling FindSquare() is 57

=> Value of num inside PrintNumbers() is 57

=> Value of num after calling FindSquare() is 57

=> Value of num before calling FindSquare() is 58

=> Value of num inside PrintNumbers() is 58

=> Value of num after calling FindSquare() is 58

=> Value of num before calling FindSquare() is 59

=> Value of num inside PrintNumbers() is 59

=> Value of num after calling FindSquare() is 59

=> Value of num before calling FindSquare() is 60

=> Value of num inside PrintNumbers() is 60

- If you try to use in, out or ref keywords with async methods, compiler will show you an error.

```
public class TestWithAsyncMethod
{
    public static async Task TestMethodAsync()
    {
        Task<int> longRunningTask = LongRunningTask(out int value);
        int result = await longRunningTask;
        Console.WriteLine("Result ");
    }

    private static async Task<int> LongRunningTask(out int value)
    {
        value = 1;
        await Task.Delay(1000);
        return value;
    }
}
```

[(parameter)] out int value

Async methods cannot have ref, in or out parameters

- Params keyword:
  - This keyword allows you to send any number of identically typed arguments as a *single logical parameter*.
  - Let's consider the following example which finds average of two numbers.

```
4 namespace c_sharp_method_passing_arguments
5 {
6     public class TestParamsKeyword
7     {
8         #region Public Methods
9         public void PrintHeader()...
10
11
12
13
14
15         public int FindAverage(int a, int b)
16         {
17             return (a + b) / 2;
18         }
19     }
20     #endregion

```

am.cs

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace c_sharp_method_passing_arguments
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             TestParamsKeyword test = new TestParamsKeyword();
11             test.PrintHeader();
12
13             Console.WriteLine($"Average = {test.FindAverage(2, 4)}");
14
15             Console.ReadLine();
16         }
17     }
18 }

```

Use of Params keyword

Average = 3



Let's say instead of two you decided to find average of more than two numbers. For that, you have to change your code but how many times you are going to change it?

So, by making changes to the method definition again and again, you can make use of **params** keyword. Let the user pass any number of arguments, there is no need to make changes to the method signature if it uses **params** keyword.

But there are some rules while using **params** keyword:

1. The parameter must be a single dimensional array.

```
public int FindAverage(params int[] numbers)
{
```

2. You can define additional parameters before using params keyword like this.

```
public int FindAverage(string str, params int[] numbers)
{
```

But if you define parameters after **params** keyword, compiler will show you an error.

```
public int FindAverage(params int[] numbers, string str)
{
```

A params parameter must be the last parameter in a formal parameter list

3. Instead of passing integers, you can also pass an array of integers or a list separated by comma to the methods containing **params** keyword.

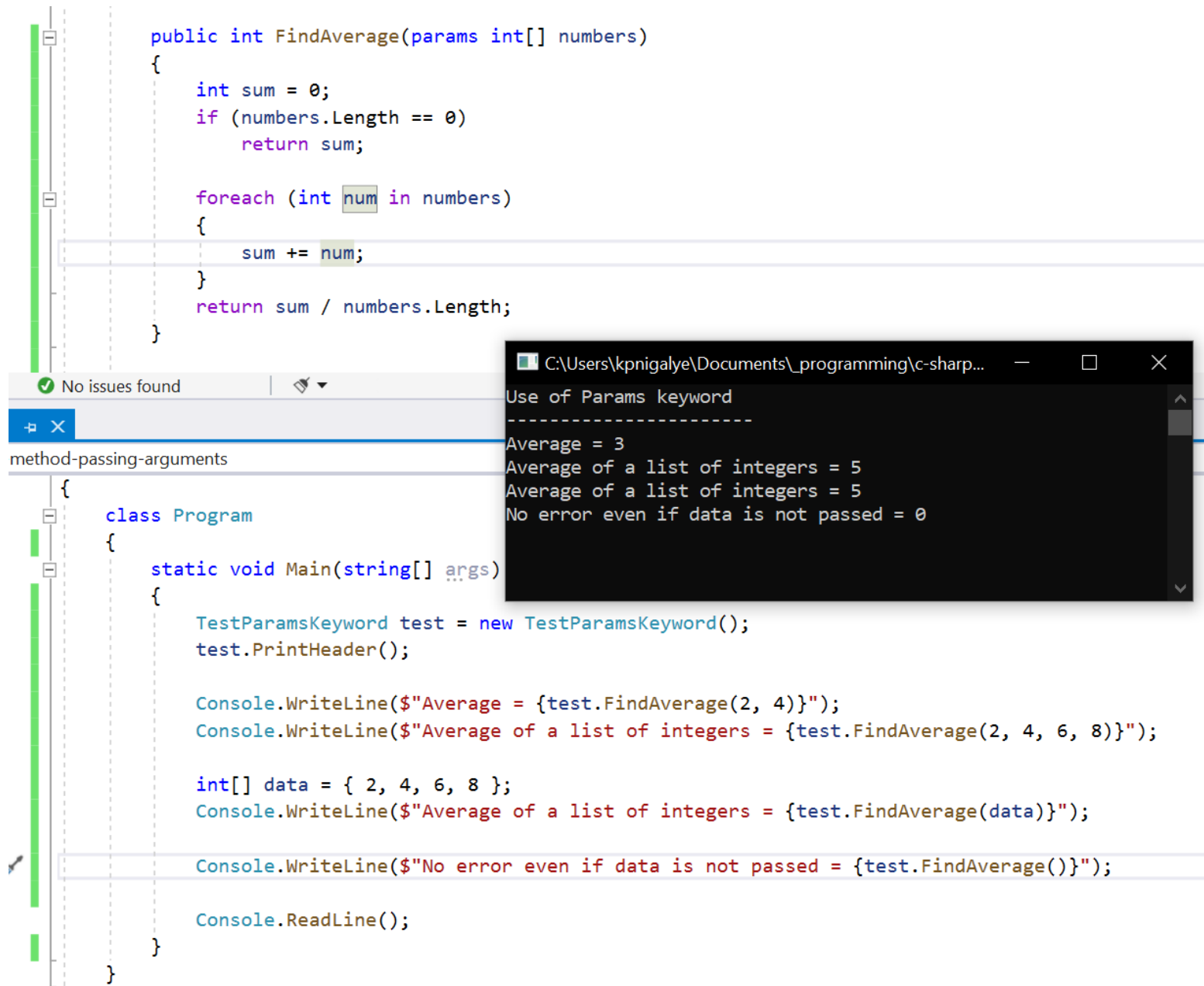
```
Console.WriteLine($"Average = {test.FindAverage(2, 4)}");
Console.WriteLine($"Average of a list of integers = {test.FindAverage(2, 4, 6, 8)}");

int[] data = { 2, 4, 6, 8 };
Console.WriteLine($"Average of a list of integers = {test.FindAverage(data)}");
```

4. Since it expects an array as an incoming parameter, even if you pass nothing as an argument to the method with **params** keyword, it will not show any compiler error.

```
Console.WriteLine($"Average of a list of integers = {test.FindAverage()}");
```

Here is the output of the whole code making use of **params** keyword.



```
public int FindAverage(params int[] numbers)
{
    int sum = 0;
    if (numbers.Length == 0)
        return sum;

    foreach (int num in numbers)
    {
        sum += num;
    }
    return sum / numbers.Length;
}
```

No issues found

```
method-passing-arguments
{
    class Program
    {
        static void Main(string[] args)
        {
            TestParamsKeyword test = new TestParamsKeyword();
            test.PrintHeader();

            Console.WriteLine($"Average = {test.FindAverage(2, 4)}");
            Console.WriteLine($"Average of a list of integers = {test.FindAverage(2, 4, 6, 8)}");

            int[] data = { 2, 4, 6, 8 };
            Console.WriteLine($"Average of a list of integers = {test.FindAverage(data)}");

            Console.WriteLine($"No error even if data is not passed = {test.FindAverage()}");

            Console.ReadLine();
        }
    }
}
```

C:\Users\kpnigalye\Documents\\_programming\c-sharp...  
Use of Params keyword  
-----  
Average = 3  
Average of a list of integers = 5  
Average of a list of integers = 5  
No error even if data is not passed = 0