



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.

```
Program.cs
1 namespace c_sharp_out_ref_keywor
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_keywor
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.

```

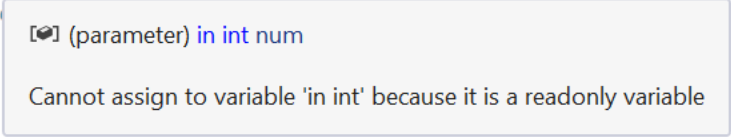
Program.cs
1
2 namespace c_sharp_out_ref_keywords
3 {
4     class Program
5     {
6         static void Main(string[] args)
7         {
8             TestOutKeyword.TestPassByReference();
9         }
10    }
11
12
TestOutKeyword.cs
1 using System;
2
3 namespace c_sharp_out_ref_keywords
4 {
5     public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

```

C:\Users\kpnigalye\Documents\_programming\c-shar...
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 arguments as a single logical parameter.
 - Let's consider the following example which adds two numbers.



```
namespace c_sharp_method_passing_arguments
{
    class Program
    {
        static void Main(string[] args)
        {
            TestParamsKeyword test = new TestParamsKeyword();
            Console.WriteLine($"Sum = {test.FindSum(5, 10)}");

            Console.ReadLine();
        }
    }
}


public class TestParamsKeyword
{
    #region Public Methods

    public int FindSum(int a, int b)
    {
        return a + b;
    }

    #endregion
}
```

The screenshot shows a Visual Studio IDE with two code files. The top file, 'keyword.cs', contains the 'Program' class with a 'Main' method that creates a 'TestParamsKeyword' object and calls its 'FindSum' method with arguments 5 and 10. The bottom file, 'method-passing-arguments.cs', contains the 'TestParamsKeyword' class with a 'FindSum' method that returns the sum of two integers. A console window titled 'C:\Users\kpnigalye\Documents\...' is overlaid on the right, displaying the output 'Sum = 15'. The status bar at the bottom of the IDE indicates 'No issues found'.

Let's say instead of two you decided to find sum of three numbers. You have to change your code like this.



```
namespace c_sharp_method_passing_arguments
{
    class Program
    {
        static void Main(string[] args)
        {
            TestParamsKeyword test = new TestParamsKeyword();
            Console.WriteLine($"Sum = {test.FindSum(5, 10, 15)}");

            Console.ReadLine();
        }
    }
}

public class TestParamsKeyword
{
    #region Public Methods

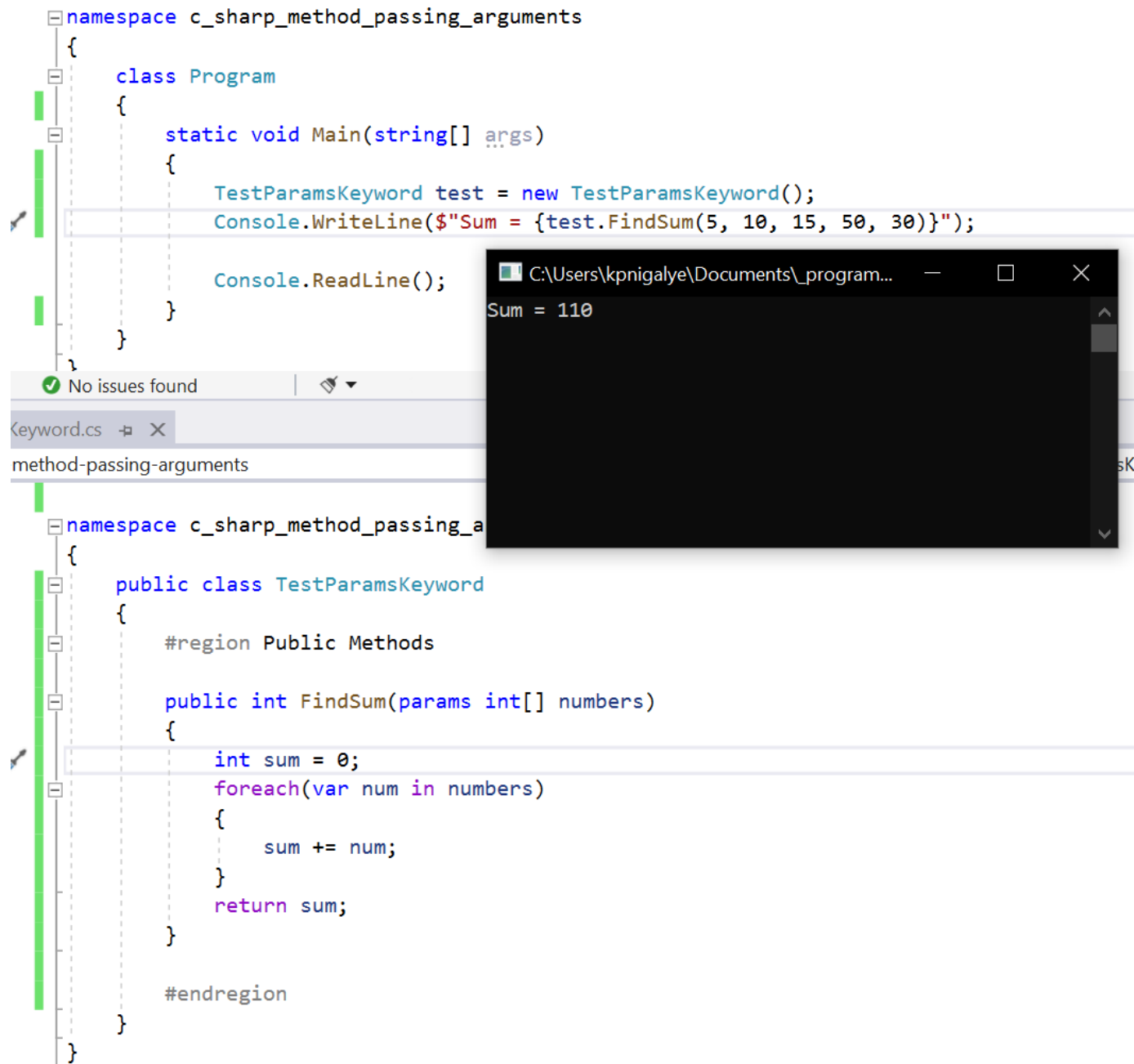
    public int FindSum(int a, int b, int c)
    {
        return a + b + c;
    }

    #endregion
}
```

Now if in future you want to find sum of more than three numbers then you will have to change the method signature again and again. Instead you can make use of params keyword. Let it be any number of method arguments, in the method signature it will be just one parameter defined with 'params' keyword.

But there are some rules while using params keyword:

- The parameter must be a single dimensional array.



```
namespace c_sharp_method_passing_arguments
{
    class Program
    {
        static void Main(string[] args)
        {
            TestParamsKeyword test = new TestParamsKeyword();
            Console.WriteLine($"Sum = {test.FindSum(5, 10, 15, 50, 30)}");

            Console.ReadLine();
        }
    }
}

namespace c_sharp_method_passing_a
{
    public class TestParamsKeyword
    {
        #region Public Methods

        public int FindSum(params int[] numbers)
        {
            int sum = 0;
            foreach(var num in numbers)
            {
                sum += num;
            }
            return sum;
        }

        #endregion
    }
}
```

Keyword.cs X

method-passing-arguments

✓ No issues found

C:\Users\kpnigalye\Documents_program... Sum = 110

- You can define additional parameters before using `params` keyword but not after it.

```

namespace c_sharp_method_passing_arguments
{
    class Program
    {
        static void Main(string[] args)
        {
            TestParamsKeyword test = new TestParamsKeyword();
            Console.WriteLine($"Sum = {test.FindSum(true, 5, 10, 15, 50, 30)}");

            Console.ReadLine();
        }
    }
}

```

```

namespace c_sharp_method_passing_arguments
{
    public class TestParamsKeyword
    {
        #region Public Methods

        public int FindSum(bool flag, params int[] numbers)
        {
            int sum = 0;
            foreach (var num in numbers)
            {
                sum += num;
            }

            return flag ? sum *= 2 : sum;
        }
    }
}

```

Console Window Output: Sum = 220

The above program sends a Boolean value before *params* keyword and if it is set to true, then doubles the sum before returning it to the caller. Instead of numbers, you can also pass a list to the methods containing *params* keyword.

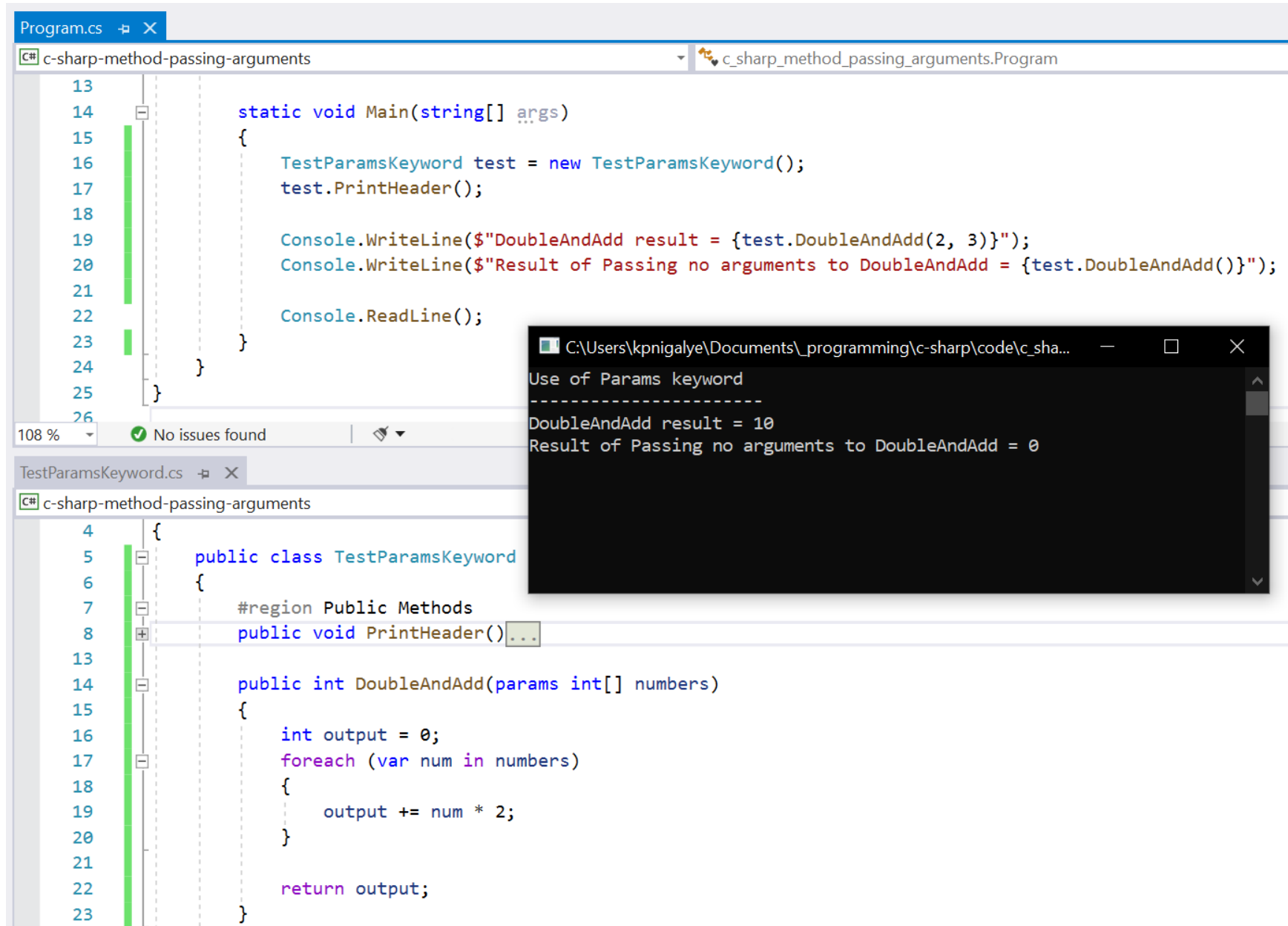
```

Console.WriteLine($"Sum of numbers = {test.FindSum(true, 5, 10, 15, 50, 30)}");
Console.WriteLine($"Sum of all numbers in the array = {test.FindSum(false, new [] { 1,3,5,7,9})}");

```

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.

In the code shown below, DoubleAndAdd method uses *params* keyword. Check both the function calls made in Main method.



```
Program.cs
c-sharp-method-passing-arguments
13
14 static void Main(string[] args)
15 {
16     TestParamsKeyword test = new TestParamsKeyword();
17     test.PrintHeader();
18
19     Console.WriteLine($"DoubleAndAdd result = {test.DoubleAndAdd(2, 3)}");
20     Console.WriteLine($"Result of Passing no arguments to DoubleAndAdd = {test.DoubleAndAdd()}");
21
22     Console.ReadLine();
23 }
24
25
26
108 % No issues found

TestParamsKeyword.cs
c-sharp-method-passing-arguments
4 {
5     public class TestParamsKeyword
6     {
7         #region Public Methods
8         public void PrintHeader()...
9
10
11
12
13
14     public int DoubleAndAdd(params int[] numbers)
15     {
16         int output = 0;
17         foreach (var num in numbers)
18         {
19             output += num * 2;
20         }
21
22         return output;
23     }
24 }
```

```
C:\Users\kpnigalye\Documents\_programming\c-sharp\code\c_sha...
Use of Params keyword
-----
DoubleAndAdd result = 10
Result of Passing no arguments to DoubleAndAdd = 0
```