PARAMETER PASSING IN C#

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/passing-reference-type-parameters

PASSING VALUE TYPES BY VALUE

A value-type variable contains its data directly as opposed to a reference-type variable, which contains a reference to its data. Passing a value-type variable to a method by value means passing a copy of the variable to the method. Any changes to the parameter that take place inside the method have no affect on the original data stored in the argument variable. If you want the called method to change the value of the parameter, you must pass it by reference, using the ref or out keyword. For simplicity, the following examples use `ref`.

Passing Value Types by Value

The following example demonstrates passing value-type parameters by value. The variable `n` is passed by value to the method `SquareIt`. Any changes that take place inside the method have no affect on the original value of the variable.

```
class PassingValByVal
{
    static void SquareIt(int x)
    // The parameter x is passed by value.
    // Changes to x will not affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(n);  // Passing the variable by value.
        System.Console.WriteLine("The value after calling the method: {0}", n);

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    The value before calling the method: 5
    The value inside the method: 25
    The value after calling the method: 5
*/
```

The variable `n` is a value type. It contains its data, the value `5`. When `SquareIt` is invoked, the contents of `n` are copied into the parameter `x`, which is squared inside the method. In `Main`, however, the value of `n` is the same after calling the `SquareIt` method as it was before. The change that takes place inside the method only affects the local variable `x`.

PASSING VALUE TYPES BY REFERENCE

The following example is the same as the previous example, except that the argument is passed as a `ref` parameter. The value of the underlying argument, `n`, is changed when `x` is changed in the method.

```
class PassingValByRef
{
    static void SquareIt(ref int x)
    // The parameter x is passed by reference.
    // Changes to x will affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(ref n);  // Passing the variable by reference.
        System.Console.WriteLine("The value after calling the method: {0}", n);

        // Keep the console window open in debug mode.
        System.Console.WriteLine("Press any key to exit.");
        System.Console.ReadKey();
    }
}
/* Output:
    The value before calling the method: 5
    The value inside the method: 25
    The value after calling the method: 25
*/
```

PASSING VALUE TYPES BY REFERENCE

In this example, it is not the value of `n` that is passed; rather, a reference to `n` is passed. The parameter `x` is not an int; it is a reference to an `int`, in this case, a reference to `n`. Therefore, when `x` is squared inside the method, what actually is squared is what `x` refers to, `n`.

SWAPPING VALUE TYPES

A common example of changing the values of arguments is a swap method, where you pass two variables to the method, and the method swaps their contents. You must pass the arguments to the swap method by reference. Otherwise, you swap local copies of the parameters inside the method, and no change occurs in the calling method. The following example swaps integer values.

C# Copy

```
static void SwapByRef(ref int x, ref int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

When you call the `SwapByRef` method, use the `ref` keyword in the call, as shown in the following example.

C# Copy

```
static void Main()
{
    int i = 2, j = 3;
    System.Console.WriteLine("i = {0}  j = {1}" , i, j);

    SwapByRef (ref i, ref j);

    System.Console.WriteLine("i = {0}  j = {1}" , i, j);

    // Keep the console window open in debug mode.
    System.Console.WriteLine("Press any key to exit.");
    System.Console.ReadKey();
}
/* Output:
    i = 2  j = 3
    i = 3  j = 2
*/
```

PASSING REFERENCE-TYPE PARAMETERS

A variable of a reference type does not contain its data directly; it contains a reference to its data. When you pass a reference-type parameter by value, it is possible to change the data pointed to by the reference, such as the value of a class member. However, you cannot change the value of the reference itself; that is, you cannot use the same reference to allocate memory for a new class and have it persist outside the block. To do that, pass the parameter using the ref or out keyword. For simplicity, the following examples use `ref`.

Passing Reference Types by Value

The following example demonstrates passing a reference-type parameter, `arr`, by value, to a method, `Change`. Because the parameter is a reference to `arr`, it is possible to change the values of the array elements. However, the attempt to reassign the parameter to a different memory location only works inside the method and does not affect the original variable, `arr`.

PASS BY REF EXAMPLE

```
class PassingRefByVal
{
    static void Change(int[] pArray)
    {
        pArray[0] = 888;  // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4};   // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr [0]);
    }
}
/* Output:
    Inside Main, before calling the method, the first element is: 1
    Inside the method, the first element is: -3
    Inside Main, after calling the method, the first element is: 888
*/
```

SWAPPING TWO STRINGS

Swapping strings is a good example of passing reference-type parameters by reference. In the example, two strings, `str1` and `str2`, are initialized in `Main` and passed to the `SwapStrings` method as parameters modified by the `ref` keyword. The two strings are swapped inside the method and inside `Main` as well.

C# Copy

```
class SwappingStrings
{
    static void SwapStrings(ref string s1, ref string s2)
    // The string parameter is passed by reference.
    // Any changes on parameters will affect the original variables.
    {
        string temp = s1;
        s1 = s2;
        s2 = temp;
        System.Console.WriteLine("Inside the method: {0} {1}", s1, s2);
    }

    static void Main()
    {
        string str1 = "John";
        string str2 = "Smith";
        System.Console.WriteLine("Inside Main, before swapping: {0} {1}", str1, str2);

        SwapStrings(ref str1, ref str2);   // Passing strings by reference
        System.Console.WriteLine("Inside Main, after swapping: {0} {1}", str1, str2);
    }
}
/* Output:
    Inside Main, before swapping: John Smith
    Inside the method: Smith John
    Inside Main, after swapping: Smith John
*/
```

SWAPPING TWO STRINGS

In this example, the parameters need to be passed by reference to affect the variables in the calling program. If you remove the `ref` keyword from both the method header and the method call, no changes will take place in the calling program.

WHAT'S THE OUTPUT

```
class Program
{
    1 reference
    static void Change(int[] pArray)
    {
        pArray[0] = 888;
        pArray = new int[5] {7, 8, 9, 10, 11};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    0 references
    static void Main()
    {
        int[] arr = {6, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr [0]);
    }
}
}
```

WHAT'S THE OUTPUT

```
class Program2
{
    1 reference
    static void Change(ref string[] pArray)
    {

        pArray[0] = "Song time";
        pArray = new string[4] { "Oompa", "loompa", "doompety", "doo" };
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    0 references
    static void Main()
    {
        string[] arr = { "I've", "got", "a perfect", "puzzle" ,"for you" };
        System.Console.WriteLine("Inside Main, before calling the method, the first element is: {0}", arr[0]);

        Change(ref arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element is: {0}", arr[0]);
    }
}
```

WHAT'S THE OUTPUT

```
7  namespace parameterPassing
8  {
       0 references
9      class Program
10     {
           0 references
11         static void Main(string[] args)
12         {
13
14             int[] studentMarks= new int[5]{23,34,54,65,73};
15             Console.WriteLine(studentMarks[3]);
16             luckDay(studentMarks);
17             Console.WriteLine(studentMarks[3]);
18             newMarks(studentMarks);
19             Console.WriteLine(studentMarks[3]);
20         }
21
22
           1 reference
23         public static void luckDay(int[] m)
24         {
25             for(int i=0; i< m.Length; i++){
26                 m[i] += 10;
27
28             }
29         }
30
           1 reference
31         public static void newMarks(int[] m)
32         {
33             int[] newGrades = new int[5] {90,98,96,95,99};
34             m = newGrades;
35         }
36
37
38     }
39 }
40
```

REF PARAMETERS VS. OUT PARAMETERS IN C#

Both ref and out are treated differently at run time and they are treated the same at compile time

Ref	Out
The parameter or argument must be initialized first before it is passed to ref.	It is not compulsory to initialize a parameter or argument before it is passed to an out.
It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method.	A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method.

EXAMPLE

```
namespace OutRefExample
{
    0 references
    class Program
    {
        1 reference
        public static string GetNextNameByOut(out int id)
        {
            id = 1;
            string returnText = "Next-" + id.ToString();
            return returnText;
        }
        0 references
        static void Main(string[] args)
{

        int i; // would have to use i=0 if using ref
        string test = GetNextNameByOut(out i);
        Console.WriteLine("Current value of integer i:" + i.ToString());
        Console.ReadLine();
}


        }
}
```