

## Day 20 – Assignment

18-Feb,2022

By [Manoj Karnatapu](#) - NBHealthCareTechnologies



### Assignment 1

Research and understand the scope of variables in C#

### Answer

#### Scope of a Variable:

A scope is a region of the program and broadly speaking there are three places, where variables can be declared:

- Inside a function or a block which is called local variables,
- In the definition of function parameters which is called formal parameters.
- Outside of all functions which is called global variables.

#### Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables:

#### Code

```
using System;

// Author : Manoj.Karnatapu
// Purpose : To find the Scope of a Variable in C#.

namespace ScopeOfVariables
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // Local Variable declaration
            int a, b;
            int c;

            // Actual initialization.
            a = 10;
            b = 20;
            c = a + b;
```

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

## Global Variables

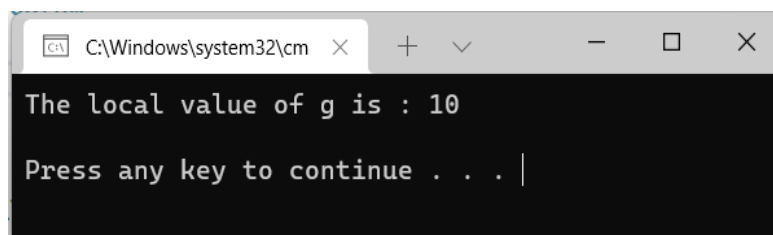
Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their type throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration

A program can have same name for local and global variables but value of local variable inside a function will take preference

## Code

```
using System;  
  
// Author : Manoj.Karnatapu  
// Purpose : To find the Scope of a Variable in C#.  
  
namespace ScopeOfVariables  
{  
    public class Program  
    {  
        int g = 20;  
        static void Main(string[] args)  
        {  
            // Local Variable declaration  
            int g = 10;  
  
            Console.WriteLine("The local value of g is : " + g);  
            Console.ReadLine();  
        }  
    }  
}
```



The screenshot shows a Windows command prompt window with the title bar 'C:\Windows\system32\cmd'. The window contains the following text:  
The local value of g is : 10  
Press any key to continue . . . |

## Assignment 2

What are Delegates in C#? Write the points discussed in the class & illustrate with a C# Code Example.

### Answer

#### Delegates in C#:

- A delegate is a type that represents references to methods with a particular parameter list and return type.
- Delegates are used to pass methods as arguments to other methods.

#### Declaration of Delegates:

Delegate type can be declared using the delegate keyword. Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

#### Syntax:

```
[modifier] delegate [return_type] [delegate_name] ([parameter_list]);
```

- A Delegate is like a function pointer.
- Using delegates, we can call (or) point to one or more methods.
- When declaring a delegate's return type & parameters must match with the methods you want to point, using delegates.
- Benefit of delegate is that, using single call from delegate all your methods pointing to delegate will be called.
- There are Two types of Delegates in C#, they are :
  - Single cast delegate
  - Multi cast delegate

### Code

```
using System;

// Author : Manoj.Karnatapu
// Purpose : Delegate Example

// For reference, check DelegatesExample in the same repository.

namespace DelegatesExample
{
    public delegate void rectDelegate(double height, double width);
    class Rectangle
    {
        // "area" method
        public void area(double height, double width)
        {
            Console.WriteLine("Area is: {0}", (width * height));
        }

        // "perimeter" method
        public void perimeter(double height, double width)
        {
            Console.WriteLine("Perimeter is: {0} ", 2 * (width + height));
        }
    }
    internal class Program
    {
        static void Main(string[] args)
```

```

{
    // creating object of class
    // "rectangle", named as "rect"
    Rectangle rect = new Rectangle();

    // these two lines are normal calling
    // of that two methods
    // rect.area(6.3, 4.2);
    // rect.perimeter(6.3, 4.2);

    // creating delegate object, name as "rectdele"
    // and pass the method as parameter by
    // class object "rect"
    rectDelegate rectdele = new rectDelegate(rect.area);

    // also can be written as
    // rectDelegate rectdele = rect.area;

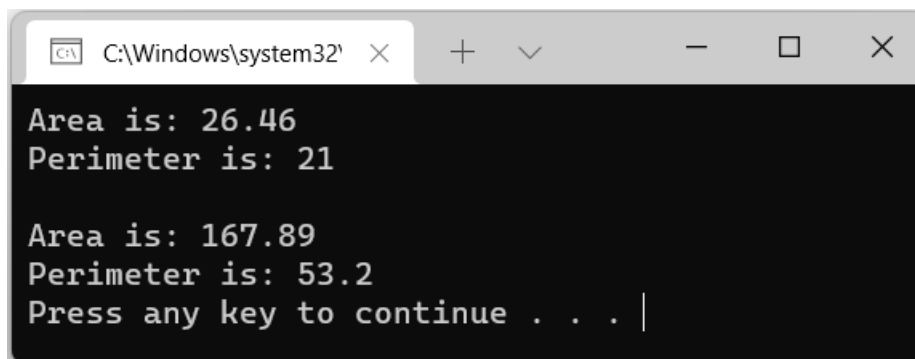
    // call 2nd method "perimeter"
    // Multicasting
    rectdele += rect.perimeter;

    // pass the values in two method
    // by using "Invoke" method
    rectdele.Invoke(6.3, 4.2);
    Console.WriteLine();

    // call the methods with
    // different values
    rectdele.Invoke(16.3, 10.3);
}
}
}

```

## Output



```

C:\Windows\system32
Area is: 26.46
Perimeter is: 21

Area is: 167.89
Perimeter is: 53.2
Press any key to continue . . . |

```

## Assignment 3

What are Nullable types in C#? Write the properties of nullable types & illustrate with a C# Code Example

### Answer

- The Nullable type allows you to assign a null value to a variable.
- Only for Reference types, we can use Nullable type. We can't use nullable for Value Types.

In order to declare a variable as a Nullable type, we place "?" symbol, adjacent to its data type.

Points to Remember :

- Nullable<T> type allows assignment of null to value types.
- ? operator is a shorthand syntax for Nullable types.
- Use **value** property to get the value of nullable type.
- Use **HasValue** property to check whether value is assigned to nullable type or not.
- Static Nullable class is a helper class to compare nullable types.

### Code

```
using System;
// Author : Manoj.Karnatapu
// Purpose : Nullable Type C# Code Example.

// For reference, check NullableTypes project in the same reference.

namespace NullableTypes
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int? firstValue = 20;
            int? secondValue = null;

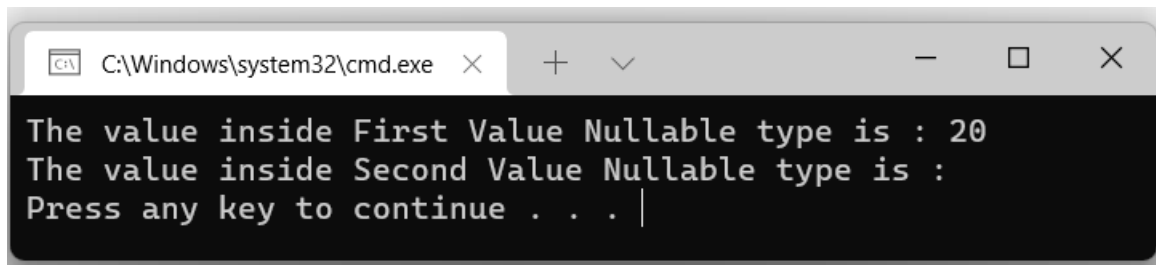
            int? result;

            result = (firstValue.HasValue) ? firstValue : null;
            Console.WriteLine("The value inside First Value Nullable type is : {0}",
result);

            result = (secondValue.HasValue) ? secondValue : null;
            Console.WriteLine("The value inside Second Value Nullable type is : {0}",
result);
            // If u find No Output displaying in the console window means, it's value is
            null i.e., unassigned.

            Console.ReadKey();
        }
    }
}
```

## Output



```
C:\Windows\system32\cmd.exe
The value inside First Value Nullable type is : 20
The value inside Second Value Nullable type is :
Press any key to continue . . . |
```

## Assignment 4

Research on out & ref parameters & illustrate with a C# Code example.

## Answer

### C# Ref & Out Keywords:

Ref and out keywords in C# are used to pass arguments within a method or function. Both indicate that an argument/parameter is passed by reference. By default parameters are passed to a method by value. By using these keywords (ref and out) we can pass a parameter by reference

### Ref Keyword:

The ref keyword passes arguments by reference. It means any changes made to this argument in the method will be reflected in that variable when control returns to the calling method.

### Out Keyword:

The out keyword passes arguments by reference. This is very similar to the ref keyword.

## Ref Vs Out

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.
Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter.	Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method.
It is not compulsory to initialize a parameter value before using it in a calling method.	A parameter value must be initialized within the calling method before its use.
When we use REF, data can be passed bi-directionally.	When we use OUT data is passed only in a unidirectional way (from the called method to the caller method).
Both ref and out are treated differently at run time and they are treated the same at compile time.	

Properties are not variables; therefore, it cannot be passed as an out or ref parameter

## Code

```
using System;
// Author : Manoj.Karnatapu
// Purpose : C# Code to illustrate, the Ref & Out keywords.

// For Reference, check RefAndOutParameters Project in the same Repository.

namespace RefAndOutParameters
{
    internal class Program
    {
        /// <summary>
        /// This is a method for Ref Keyword Example
        /// </summary>
        /// <param name="id"> ref id</param>
        /// <returns>NextId</returns>
        public static string NextNameByRef(ref int id)
        {
            string returnText = "Next-" + id.ToString();
            id += 1;
            return returnText;
        }
        /// <summary>
        /// This is a Method for Out Keyword Example
        /// </summary>
        /// <param name="id">out id</param>
        /// <returns>NextId</returns>
        public static string NextNameByOut(out int id)
        {
            id = 1;
            string returnText = "Next-" + id.ToString();
            return returnText;
        }
        /// <summary>
        /// Main Method, Code Starting point.
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            Console.WriteLine("\n -----**** Ref Keyword Output ****-----");

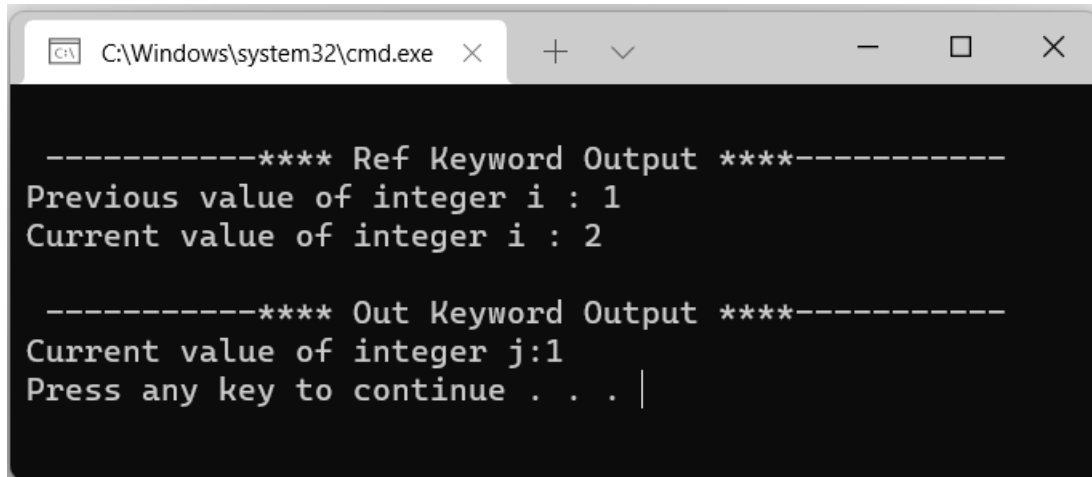
            int i = 1;
            Console.WriteLine("Previous value of integer i : " + i.ToString());
            string testRef = NextNameByRef(ref i);
            Console.WriteLine("Current value of integer i : " + i.ToString());

            Console.WriteLine("\n -----**** Out Keyword Output ****-----");

            int j;
            string testOut = NextNameByOut(out j);
            Console.WriteLine("Current value of integer j:" + j.ToString());

            Console.ReadKey();
        }
    }
}
```

## Output

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Windows\system32\cmd.exe' and standard window controls. The command prompt has a black background with white text. The output consists of two sections separated by dashed lines. The first section is titled 'Ref Keyword Output' and shows the 'Previous value of integer i : 1' and 'Current value of integer i : 2'. The second section is titled 'Out Keyword Output' and shows 'Current value of integer j:1' followed by 'Press any key to continue . . . |' with a vertical cursor.

```
-----**** Ref Keyword Output ****-----  
Previous value of integer i : 1  
Current value of integer i : 2  
  
-----**** Out Keyword Output ****-----  
Current value of integer j:1  
Press any key to continue . . . |
```

**The End**