

DAY20 ASSIGNMENT
BY
PAVAN KUMAR (18-02-2022)

Q1). Research and understand the scope of variables in C#

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.

GLOBAL VARIABLE:

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

Local Variables:

These 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:

Q2) What are Delegates in C#? Write the points discussed in the class & illustrate with a C# Code Example.

DELEGATES:

- 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.
- A Delegate is like a function pointer.
- Using delegates, we can call (or) point to one or more methods.

There are Two types of Delegates in C#, they are:

- Single cast delegate
- Multi cast delegate

CODE:

```
using System;
namespace Delegates Example
{
    /// <summary>
    /// DONE BY: PAVAN
    /// PURPOSE: DELEGATES EXAMPLE PROGRAM
    /// </summary>
    /// <param name="height"></param>
```

```
/// <param name="width"></param>
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();

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

        rectdele += rect.perimeter;

        // by using "Invoke" method
        rectdele.Invoke(4, 5);
        Console.WriteLine();

        // call the methods with
        // different values
        rectdele.Invoke(16, 10);
    }
}
}
```

OUTPUT:

```
C:\WINDOWS\system32\cmd.exe

Area is: 20
Perimeter is: 18

Area is: 160
Perimeter is: 52
Press any key to continue . . .
```

Q3). Research on out & ref parameters & illustrate with a C# Code example.

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 KEYWORD	OUT KEYWORD
1). The parameter or argument must be initialized first before it is passed to ref.	1). It is not compulsory to initialize a parameter or argument before it is passed to an out.
2). When we use REF, data can be passed bidirectionally.	2). When we use OUT data is passed only in a unidirectional way.
3). It is not compulsory to initialize a parameter value before using it in a calling method.	3). A parameter value must be initialized within the calling method before its use.

CODE:

```
using System;

namespace RefAndOutParameters
{
    /// <summary>
    /// DONE: PAVAN
    /// PURPOSE: REF AND OUT KEYWORDS
    /// </summary>
    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;
}

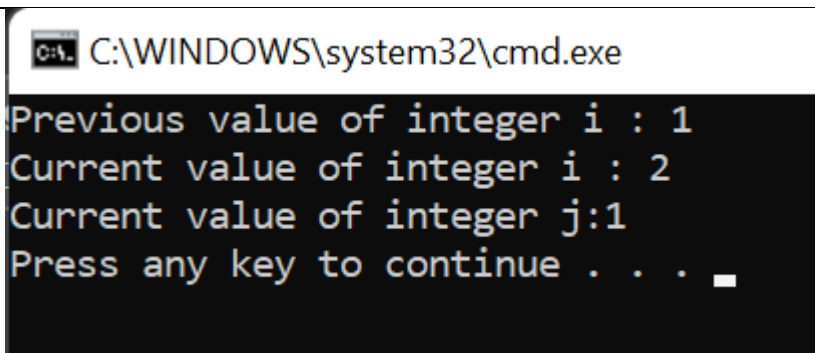
public static string NextNameByOut(out int id)
{
    id = 1;
    string returnText = "Next-" + id.ToString();
    return returnText;
}
/// <summary>
/// Main Method, Code Starting point.
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    int i = 1;
    Console.WriteLine("Previous value of integer i : " + i.ToString());
    string testRef = NextNameByRef(ref i);
    Console.WriteLine("Current value of integer i : " + i.ToString());

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

    Console.ReadKey();
}
}
}

```

OUTPUT:



The screenshot shows a Windows command prompt window with the title bar 'C:\WINDOWS\system32\cmd.exe'. The output of the program is displayed in a monospaced font:

```

Previous value of integer i : 1
Current value of integer i : 2
Current value of integer j:1
Press any key to continue . . .

```

Q4). What are nullable types in C# WACP to illustrate nullable types Write some properties of nullable types

NULLABLE TYPES:

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.

CODE:

```
using System;
namespace NullableTypes
{
    /// <summary>
    /// DONE BY: PAVAN
    /// PURPOSE: NULLABLE TYPE
    /// </summary>
    internal class Program
    {
        static void Main(string[] args)
        {

            int? firstValue = 20;
            int? secondValue = null;

            int? result;

            result = (firstValue.HasValue) ? firstValue : null;

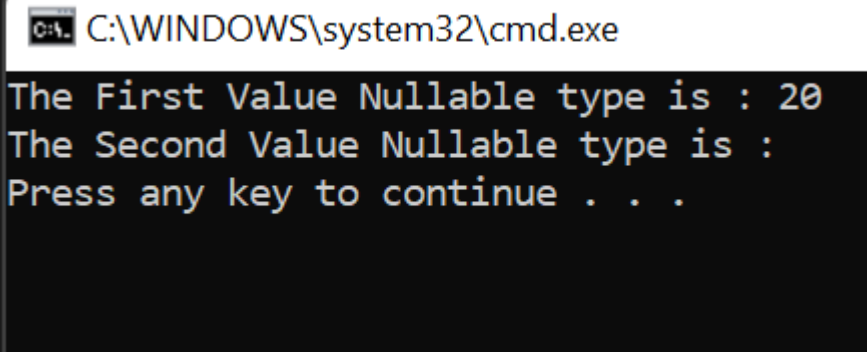
            Console.WriteLine("The First Value Nullable type is: {0}", result);

            result = (secondValue.HasValue) ? secondValue : null;

            Console.WriteLine("The Second Value Nullable type is: {0}", result);

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

OUTPUT:



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