

**Day 20 Assignment**

**By**

**Narala Praveen**

**18-02-2022**

## Question 1:

### Research and understand scope of variables in c#?

#### Scope of Variables:

The part of the program where a particular variable is accessible is termed as the scope of that variable. A variable can be defined in a Class, Method, loop etc.

Scope rules of variables can be divided into three categories:

- . Class level scope
- . Method level scope
- .Block level scope

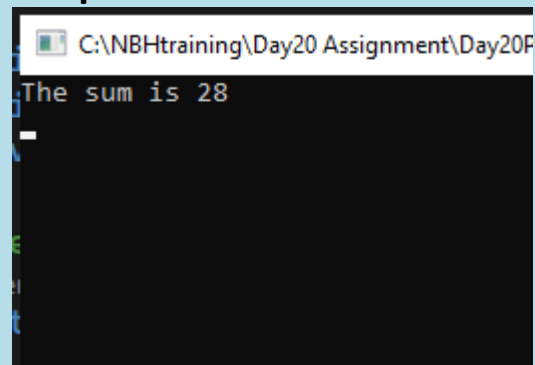
#### Class level scope Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Project1
{
    //Author:Narala Praveen
    //Purpose :Scope of Variables
    class ClassLevelscope
    //class scope starts
    {
        public static int a = 10; //class level variables or class members
        public static int b = 18;
        private static int sum = 0;

        //Method scope starts
        static void Main(string[] args)
        {
            sum = a + b;
            Console.WriteLine(sum);
            Console.ReadLine();
        } //Method scope ends
    } //class scope ends
}
```

#### Output:



**Method Level Scope:** If the variable declared in Side a Method is called as Method level scope. It can't be accessed out Side of the method.

**Example:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Methodscope
{
    class MethodLevelscope
    {
        public void Display()
        {
            int i = 10;
            Console.WriteLine(i); //with in method declared and accepted in side
it
        }
        static void Main(string[] args)
        {
            Console.WriteLine(i); //out side the method no accessible
            Console.ReadLine();
        }
    }
}
```

**Block Level Scope:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlockleveScope
{
    class BlockLevelscope
    {
        static void Main(string[] args)
        {
            for ( int i = 0; i <= 5; i++) ; // int i scope is with in the for
block
            Console.WriteLine(i);
            Console.ReadLine();
        }
    }
}
```

## Question 2:

What are delegates in c#

Write the points discussed about delegates in the class WACP to illustrate the usage of delegates?

**Delegates:** A Delegate is like a function pointer. A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type.

**Points:**

1. A delegate is like a function pointer.
2. Using delegate we can call or point to one or more methods.
3. When declaring a delegate return type and parameters must match with methods you want to point using the delegate.
4. Benefits of delegate is that we call the method with single and we can also remove method easily.
5. There are two types of Delegates.
  - A. Single cast delegate –single method.
  - B. Multi-Cast delegate – more than two methods.

## Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Project2
{
    //Author :Narala Praveen
    //Purpose:code for delegates

    public delegate void myfact(int n);

    internal class Program
    {
        /// <summary>
        /// This method is used for finding factorial
        /// </summary>
        /// <param name="n"></param>
        public static void Factorial(int n)
        {
            int fact = 1;
            for (int i = 1; i <= n; i++)

                fact *= i;
        }
    }
}
```

```

        Console.WriteLine(fact);

    }
    /// <summary>
    /// This method is used for finding factors
    /// </summary>
    /// <param name="n"></param>
    public static void Factors(int n)
    {

        for (int i=1;i<=n;i++)
        {

            if(n%i==0)

                Console.WriteLine(i);

        }

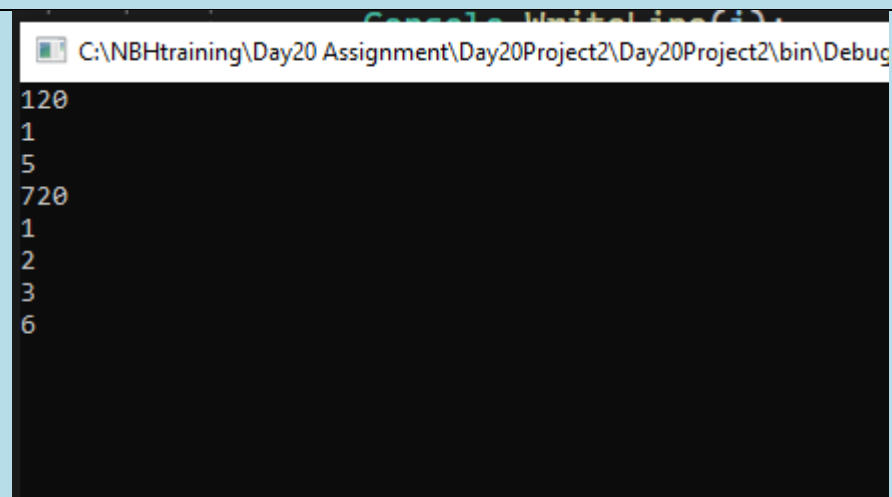
    }

    static void Main(string[] args)
    {
        myfact mf = new myfact(Factorial);
        mf += Factors;
        mf(5); //Delegate
        mf(6);

        Console.ReadLine();

    }
}

```



```

C:\NBHtraining\Day20 Assignment\Day20Project2\Day20Project2\bin\Debug
120
1
5
720
1
2
3
6

```

### Question 3:

What are nullable type in c#

WACP to illustrate nullable types

Write some properties of nullable types (like Hasvalue)

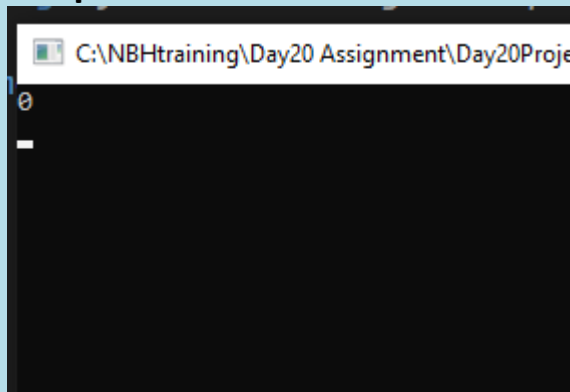
**Nullable Type:** In C# the compiler does not allow you to assign a null value to a variable, so a special feature added to c# which allows you to assign null value i.e nullable types, so the nullable type allows you to assign a null value to a variable.

#### Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Project3
{
    //Author:Narala Praveen
    //Purpose:To illustrate nullable types
    internal class Program
    {
        static void Main(string[] args)
        {
            byte? input = null;
            Console.WriteLine(Convert.ToInt16(input));
            Console.ReadLine();
        }
    }
}
```

#### Output:



#### Properties:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

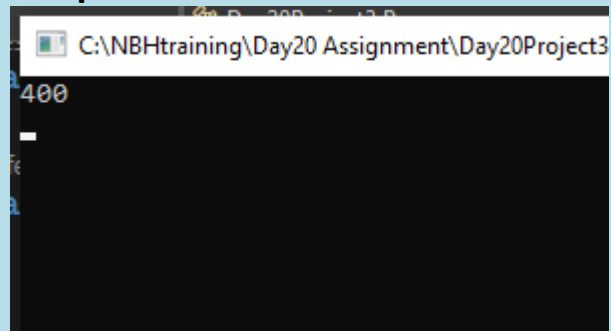
namespace Day20Project3
```

```
{ //Author:Narala Praveen
  //Purpose:To illustrate nullable types
  internal class Program
  {
      static void Main(string[] args)
      {
          Nullable <int> n = 20;

          if(n.HasValue)

              Console.WriteLine(n*n);
          else
              Console.WriteLine("no value");
          Console.ReadLine();
      }
  }
}
```

## Output:



#### Question 4:

Research and write about out and ref parameters  
WACP to illustrate them?

**Out Parameter:** out parameter is used when a method returns multiple values. When a parameter passes with the out keyword/parameter in the method , then that method works with the same variable value that is passed in the method call. If variable value changes , the method parameter value also changes.

#### Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Project4
{
    //Author:Narala Praveen
    //Purpose:To illustrate out and ref parameters
    internal class Program
    {
        static void Main(string[] args)
        {
            int i;
            int j;
            int k;
            Multiplication(out i, out j, out k); ;
            Console.WriteLine("First number is {0}",i);
            Console.WriteLine("second number is {0}", j);
            Console.WriteLine("product is {0}", k);
            Console.ReadLine();
        }
        public static void Multiplication(out int i,out int j,out int k)
        {
            i = 10;
            j=4;
            k = i * j;
        }
    }
}
```

#### Output:



```
C:\NBHtraining\Day20 Assignment\Day20Project4\Day20Project4\Program.cs
First number is 10
second number is 4
product is 40
-
```

### Ref parameter:

The ref key word is used for passing or returning references of a values to or from methods. Basically, it means that any change made to a value that is passed by reference will reflect this change since you are modifying the value at the address and not just the value . it can be implemented in the following cases:

- a. To pass an argument to a method by its reference .
- b. To define a method signature to return a reference variable.
- c. To declare a structure as a ref structure
- d. As local reference.

### Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Day20Projectref2
{
    //Author:Narala Praveen
    //Purpose:To illustrate ref parameter
    internal class Program
    {
        static void Main(string[] args)
        {
            int i = 12;
            int j = 15;
            Console.WriteLine("initially i is {0}", i);
            Console.WriteLine("initially j is {0}", j);

            AddValue(i); // normal method
            Console.WriteLine("value of i after addition operation is {0}", i);

            Subtract(ref j); // ref declaration
            Console.WriteLine("value of j after subtraction operation is {0}", j);

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

```
    }  
    public static void AddValue(int i)  
    {  
        i += 12;  
    }  
    public static void Subtract(ref int j)  
    {  
        j -= 12;  
    }  
}
```

C:\NBHtraining\Day20 Assignment\Day20Projectref2\Day20Projectre

```
initially i is 12  
initially j is 15  
value of i after addition operation is12  
value of j after subtraction operation is 3  
-
```