Introduction .NET Framework

.NET 4.5

About the Author

- Created By: Mohammad Salman
- Experience: 12 Years +
- Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Questions?

Reference

Try it Out

Informative Slide

Mandatory Slide

Welcome Break

.NET 4.0 Session Overview

Objectives:

- After completing this session, you will be able to:.
- Describe .NET and its framework.
- Identify Framework classes and CLR.

What is .NET

- The .NET Framework is an integral Windows component that supports building and running desktop applications and Web services.
- It provides a managed execution environment, simplified development and deployment, and support for a wide variety of programming languages.
- Two key components are the Common Language Runtime (CLR), which manages memory, code execution, and other system services; and the .NET Framework Class Library, which is a collection of reusable types you can use to develop your applications.

What is .NET

- .NET Framework provides a comprehensive and consistent programming model for building applications that have visually stunning user experiences and seamless and secure communication.
- Microsoft Visual Studio is a powerful IDE that ensures quality code throughout the entire application lifecycle, from design to deployment

NET Framework Version History

Version	Version Number	Release Date	Visual Studio	Operating System
1.0	1.0.3705.0	2002-02-13	VS.NET	Windows 98
1.1	1.1.4322.573	2003-04-24	VS.NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	VS.NET 2005	Windows Server 2003 R2
3.0	3.0.4506.30	2006-11-06	VS.NET 2005	Vista, Server 2008
3.5	3.5.21022.8	2007-11-19	VS.NET2008	Windows 7, Server 2008 R2
4.0	4.0.30319.1	2010-04-12	Visual Studio 2010	Windows 7, Server 2008 R2

Common Language Runtime

- The Common Language Runtime (CLR) is a core component of Microsoft's .NET initiative. It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code. In the CLR, code is expressed in a form of byte code called the Common Intermediate Language (CIL, previously known as MSIL—Microsoft Intermediate Language).
- Developers using the CLR write code in a language such as C# or VB.NET. At compile time, a .NET compiler converts such code into CIL code. At runtime, the CLR's just in time compiler converts the CIL code into code native to the operating system.

Framework Class Library

- Dot Net Framework class library is a collection of classes which allows developers to access all the functionality exposed by CLR
- This is the foundation upon which, ASP.NET applications, components and controls are built.
- The Dot Net Framework contains many in-built classes, that the developer can use when building an application.
- Dot Net framework class library is very efficient in providing supporting libraries for any kind of application in the world which developers can complete application development in a rapid manner

Framework Class Library

- There are almost 13,000 classes in the Dot Net Framework. Microsoft has divided those classes into separate namespaces in order to bind them into the Framework. All related classes could be used through those namespaces
- Using Namespaces in C# files:

```
using System;
using MyClass = MyCompany.SampleNameSpace;
```

- Using Namespaces in VB files:

```
Imports System;
Imports MyClass = MyCompany.SampleNameSpace
```

Common Type System

- The Common Type System (CTS) is a standard that specifies how Type definitions and specific values of Types are represented in computer memory.
- It is intended to allow programs written in different programming languages to easily share information.
- establish a framework that helps enable cross-language integration, type safety, and high performance code execution
- Provides an object oriented model that supports the complete implementation of many programming languages.

Common Language Specification

- A set of base rules to which any language targeting the CLI should conform in order to interoperate with other CLS-compliant languages.
- The CLS rules define a subset of the Common Type System.
- The Common Language Infrastructure (CLI) is an open specification developed by Microsoft that describes the executable code and runtime environment that form the core of the Microsoft .NET Framework.
- The specification defines an environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures.

.

Memory Management

- Common language runtime provides the automatic memory management. This feature is available during Managed Execution.
- The garbage collector manages the allocation and release of memory for an application.
- Developers need not write code to perform memory management tasks when they develop managed applications.
- Automatic memory management takes care of problems such a memory leak, or attempting to access memory for an object that has already been freed.

.

Why .NET

- Significantly faster time-to-market
- Performance
- User experience
- Developer productivity
- Innovation potential

.

.NET Case Studies

CoreLab PARTNERS

SDC
Skandinavisk Data Center A/S

Telefónica

rackspace

ORION
LIGHT YEARS AHEAD.

K2

unx[ ]

RAH

.NET Case Studies

- **Switching from Java to .NET**

  After an unsuccessful attempt to develop a mission-critical imaging application in Java, CoreLab moved its development to the .NET Framework, using the Visual Studio IDE and Visual Studio Team Foundation Server

- **Delivering applications quickly**

  Geico needed to develop its GloveBox app for Windows Phone 7 quickly to debut at the then-imminent launch of the Microsoft phone. The company met these goals and more through its use of the.NET Framework.

.NET Case Studies

- Telefonica used the .NET Framework to launch the developer platform BlueVia.
- SDC decided to move its core banking system from a mainframe to a more modern platform. Although the company had previously used Java, SDC successfully moved its mission-critical system to the.NET Framework.
- Rackspace used the .NET Framework to create a toolkit that simplifies cloud development.
- Orion Energy Systems used the .NET Framework to develop a cloud app for customers to monitor their energy consumption

.NET Case Studies

- UNX used the .NET Framework to build a desktop app for institutional investors and brokers.
- Royal Adelaide Hospital used the .NET Framework to increase its operation visibility and improve patient management and care

Q & A

- **Allow time for questions from participants.**

References

- **For More information visit:**
- **http://www.microsoft.com/net/casestudies**
- **http://msdn2.microsoft.com/**

End of the Session

- **You have completed this Session of .NET 4.0**

Overview of C# .NET 4.0

C# 4.0

About the Author

- Created By: Mohammad Salman
- Experience: 12 Years +
- Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Questions?

Reference

Try it Out

Informative Slide

Mandatory Slide

Welcome Break

General Structure of a C# Program

```
using System;
namespace YourNamespace
{
    class YourClass
    {
    }
    struct YourStruct
    {
    }
    interface IYourInterface
    {
    }
    delegate int YourDelegate();

    enum YourEnum
    {
    }
    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }
    class YourMainClass
    {
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

C# Session : Overview

Introduction:

This session gives an overview of the fundamentals of C# language including statements, identifiers, keywords, variables, and a sample C# program.

C# Session: Objectives

Objectives:

After completing this session, you will be able to:

- Identify Keywords
- List the keywords in C#
- Define Identifiers
- Define Statements
- Define and apply variables in C#
- Describe Console Application

Keywords

Keywords

- Keywords are special words built into C# language and are reserved for specific use.
- Keywords provide the concrete syntax which the compiler uses to interpret the expression.
- Keywords cannot be used for naming any classes, methods, and variables.
- The compiler uses the keywords to identify the structure and organization of the code.

List of Keywords

abstract as	base bool break byte	case catch char checked class const continue	decimal default delegate do double
else enum event explicit extern	false finally fixed float for foreach	goto	if implicit in int interface internal is
lock long	namespace new null	object operator outo override	params private protected public
readonly ref return sbyte sealed	this throw true try typeof	uint ulong unchecked unsafe ushort using	short sizeof stackallac static string struct switch
		virtual void	

Identifiers - Overview

- Identifiers are names used to denote variables, constants, types, methods, objects, and so on.
- Identifiers should be meaningful for their intended purpose and should not be arbitrary.
- Normally, it is not permitted to use keywords as identifiers, unless they are prefixed by an @ symbol.

Identifiers: Naming Convention

An identifier:

- must be a whole word.
- must begin with a letter or an underscore.
- can be any combination of letters, numbers, and underscores.
- should not start with a number.

Identifiers - Examples

Valid identifiers in C# are as follows:

- Hello
- hello
- H_ello
- HelloWorld
- _token
- @override

Invalid identifiers in C# are as follows:

- 1twothree
- Decimal

Statements

- Statements are program instructions.
- Statements are executed in sequence.
- It controls the flow of a program.

A statement can:

- declare a variable or constant
- call a method
- create an object
- assign a value to a variable, property, and so on.

Types of Statements

Following are the different types of statements:

- Selection statements
- Jump statements
- Iteration statements
- Exception handling statements
- Checked and Unchecked
- Fixed statements

Variables

- Variables represent storage locations.
- Each variable has a type that determines the values to be stored in the variable.
- The value of a variable can be assigned and that value can also be changed programmatically at any point.

Variables - Declaration

- In C# all variables must be declared before they are used.
- In the declaration, you must specify a data type and a name for the variable so that memory can be set aside for the variable.
- Syntax for variable declaration:

  <data type> <name>;

Sample:

```
int  sum = 42;
```

Variables: Naming Rules

- In C#, variable names must adhere to the following rules:
- The name can contain letters, digits, and the underscore character (_).
- The first character of the name must be a letter. The underscore is also a legal first character, but is not recommended.
- C# is case-sensitive.
- count and Count refer to two different variables.
- C# keywords cannot be used as variable names.
- The variable name cannot contain illegal characters.

Variables - Examples

Valid variables:

- intTotal
- Y2x5
- STRFirstName
- _LastName (Legal, but not advised)

Invalid Variables:

- strFirst#Name: Contains illegal character
- Double: It is a c# keyword
- 9byte: First character is a digit

Type inference

- Type inference makes use of the var keyword.
- The syntax for declaring the variable changes somewhat. The
- compiler “infers” what type the variable is by what the variable is initialized to
- Example: int a = 0; becomes var a= 0;

General Structure of a C# Program

```
using System;
namespace YourNamespace
{
    class YourClass
    {
    }
    struct YourStruct
    {
    }
    interface IYourInterface
    {
    }
    delegate int YourDelegate();

    enum YourEnum
    {
    }
    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }
    class YourMainClass
    {
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

Console Applications

- C# can be used to create applications that accepts input and display output at the command line console.
- These applications are ideal for studying C# development because the user interface is so simple.
- Console applications are also very useful for utility programs that require little or no user interaction.

Sample Program

- This is the Program for displaying the both the variables (local and global).
- Two variables are declared within the class. The same two variables declared within the main program.
- This method displays both the variable values.

```
using System;
namespace Cognizant.Academy.Sample
{
    class Sample
    {
        static int A = 50;
        static int B = 70;

        public static void Main()
        {
            int A = 200;
            int B = 500;
            Console.WriteLine("Local variable A is : " + A);
            Console.WriteLine("Local variable B is  : " + B);
            Console.WriteLine();
        }
    }
}
```

Tools

- Visual Studio 2017
- OR
- VS Code

Q & A

- **Allow time for questions from participants.**

Test Your Understanding

- What are the different types of statements and its syntax?
- How do you give valid names in C#? Give some examples.
- List some of the keywords in C#.
- What is an Identifier?
- Can a variable name contain special characters?

C# Session: Summary

- Here's a brief recap of this session:
- Statements:
- Statements are program instructions.
- Statements are executed in sequence.
- Identifiers: Identifiers are names used to denote variables, constants, types, methods, objects, and so on.
- Keywords:
- Keywords are special words built into the C# language and are reserved for specific use.
- Cannot use them for naming your classes, methods, and variables.

C# Session : Summary (Contd.)

- Variables: Variables represent storage locations. Every variable has a type that determines the values to be stored in the variable.
- Console applications: C# can be used to create applications that take input and display output at the command line console.

End of the Session

- **You have completed this Session of C#4.0**

Session : Statements in C#

About the Author

- Created By: Mohammad Salman
- Experience: 12 Years +
- Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Questions?

Reference

Try it Out

Informative Slide

Mandatory Slide

Welcome Break

C# Session 03: Overview

Introduction

- This session gives an overview of the primitive data
- types and control flow statements available in C# 4.0.

Escape Sequence

- \' Single quotation mark
- \" Double quotation mark
- \\ Backslash
- \0 Null
- \a Alert
- \b Backspace
- \f Form feed
- \n Newline
- \r Carriage return
- \t Tab character
- \v Vertical tab

C# Session 03: Objectives

Objectives:

- After completing this session, you will be able to:
- Identify Data types
- Describe Boxing and Unboxing
- Define Operators
- List various Control Flow Statements

Data Types

- Data types are the basic building blocks of any language.
- Data types can be described as being either:
- A built-in data type, such as an int or char.
- A user-defined data type, such as a class or interface.
- Data types can also be defined as being either:
- Value Type
- Reference Type

C# Data Types (Contd.)

- The information stored in a type can include the following:
- The storage space that a variable of the type requires.
- The maximum and minimum values that it can represent.
- The members (methods, fields, events, and so on) that it contains.
- The base type it inherits from.
- The location where the memory for variables will be allocated at run time.
- The kinds of operations that are permitted.

C# Data Types (Contd.)

Built-in Types:

- Integer, Floating point values, Boolean Expressions, text characters, decimal values and other types of data.
- Custom Types
- Strut, class, interface, enum
- Each type in the CTS is defined as either a value type or a reference type.
- This includes all custom types in the .NET Framework class library and also your own user-defined types.

C# Data Types (Contd.)

- Types that you define by using the struct keyword are value types; all the built-in numeric types are structs.
- Types that you define by using the class keyword are reference types.
- Reference types and value types have different compile-time rules, and different run-time behavior.

C# Data Types (Contd.)

Value Types:

- Value types derive from System.ValueType, which derives from System.Object.
- Types that derive from System.ValueType have special behavior in the CLR. Value type variables directly contain their values, which means that the memory is allocated inline in whatever context the variable is declared.
- There is no separate heap allocation or garbage collection overhead for value-type variables.

Escape Sequence

- \’ Single quotation mark
- \” Double quotation mark
- \\ Backslash
- \0 Null
- \a Alert
- \b Backspace
- \f Form feed
- \n Newline
- \r Carriage return
- \t Tab character
- \v Vertical tab

C# Data Types (Contd.)

Reference Types:

- A type that is defined as a class, delegate, array, or interface is a reference type.
- At run time, when you declare a variable of a reference type, the variable contains the value null until you explicitly create an instance of the object by using the new operator.

C# Boxing & UnBoxing

Boxing:

- Converting value types to reference types is known as boxing. As can be seen in the example below, it is not necessary to tell the compiler an INT32 is boxed to an object, because it takes care of this itself.
- Int32 x = 10;
- object o = x ; // Implicit boxing
- Console.WriteLine("The Object o = {0}",o); // prints out 10

C# Boxing & UnBoxing

UnBoxing:

- The following example intends to show how to unbox a reference type back to a value type. First an Int32 is boxed to an object, and then it is unboxed again. Note that unboxing requires explicit cast.
- Int32 x = 5;
- object o1 = x; // Implicit Boxing
- x = (int) o1; // Explicit Unboxing

C# Operators

- An operator is a program element that is applied to one or more operands in an expression or statement.
- Operators that take one operand are referred to as unary operators. For example, the increment
- operator (++) .
- Operators that take two operands are referred to as binary operators. For example such as arithmetic operators (+,-,*,/),.

C# Operators (cont)

- The sole ternary operator in C# is the conditional operator (?:) and takes three operands.
- C# has Primary Operators, Unary Operators, Multiplicative Operators, Additive Operators, Shift Operators, Relational and Type Operators, Equality Operators, Assignment ,Operator, Logical, Conditional, and Null Operator.

Control Flow Statements

Control flow statements control the execution path of the program.

- Selection statements:
- if
- switch
- Iteration statements:
- For
- goto

Control Flow Statements

Control flow statements control the execution path of the program.

- foreach
- while
- do while
- Jump statements:
- break Statement
- continue Statement
- goto

if Statement

- A conditional statement evaluates a condition and returns a value of either true or false.
- Syntax:

```
if ( boolean-expression) embedded-statement
if ( boolean-expression) embedded-statement
else
embedded-statement
```

Example:

```
int input = 1;
if (input == 1)
{
    System.Console.WriteLine("Input equals 1");
}
else
{
    System.Console.WriteLine("Input does not equals 1");
}
```

Switch Statement

- The switch statement allows you to compare the expression to one or many constant expressions and if a match is found, then the code is executed.
- You can also have a default block of code that will execute if no matches are found.
- Switch statement must have a break statement to come out of the loop, if the condition is true.

Switch Statement: Sample

```
static void Main(string[] args)
{
    int x = 29;
    string Message = "";
    switch (x)
    {
        case 27:
            Message = "Codereference is cool";
            Console.WriteLine(Message);
            break;
        case 28:
            Message = "codereference is The reference Site";
            Console.WriteLine(Message);
            break;
        case 29:
            Message = "Codereference helps me at work";
            Console.WriteLine(Message);
            break;
        case 30:
            Message="codereference is bookmarked";
            Console.WriteLine(Message);
            break;


    }
```

for Statement

- The for statement evaluates a sequence of initialization expressions and while a condition is true, it repeatedly executes an embedded statement and evaluates a sequence of iteration expressions.
- Another unique characteristic of a for loop is that it maintains an iterator value (a counter). There are three basic parts to a “for” loop:
- Initial value
- Conditional expression
- Change value

for Statement: Sample

- **Syntax of for loop is:**

```
for(initial_value;conditional_expression;change_value)
  {
      // Code to execute
  }
```

- **So if you want to loop 10 times, your code might look like:**

```
for (int x = 1; x < 10; x++)
{
    Console.WriteLine("C# is a powerfull language",x);

}
```

for Statement: Sample (Contd.)

- **Any of the three parts of the for loop (mentioned earlier) can be left blank, this will result in a infinite loop.**

```
for (; ; )
{
    Console.WriteLine("C# is a powerfull language");

}
```

foreach Statement

- The “foreach” statement enumerates the elements of a collection and for each element in the collection it executes an embedded statement.
- In this example the foreach statement is used to loop through the items of a hashtable.

```
static void Main(string[] args)
{
    Hashtable htcountries = new Hashtable();
    htcountries.Add("CA","Canada");
    htcountries.Add("USA","UnitedStates");
    htcountries.Add("CHI","China");

    foreach (string country in htcountries.Keys)
    {
        Console.WriteLine("Country:" + htcountries[country]);

    }
    Console.ReadLine();


}
```

foreach Statement

Result:

- Country: United States
- Country: Canada
- Country: China

while Statement

- The while statement is a looping statement that will continue to execute as long as the condition evaluates to true.
- The condition is evaluated before entering the code block.

Example:

```
int x = 10;

while (x < 20)
{
    Console.WriteLine(x);
    x++; //Increment the value of x
}
```

Output:

```
10
12
13
14
15
16
17
18
19
```

do while Statement

- The do-while is similar in nature to the while statement except that the block of code is guaranteed to execute at least one.
- The code will execute at least once because the conditional statement for the loop is evaluated at the end of the loop.

Example:

```
int x = 10;
do
{
    Console.WriteLine(x);
    x++; //Increment the value of x
}
while (x < 20);
```

Output:

```
10
12
13
14
15
16
17
18
19
```

break Statement

- Jump statements in C# are break, continue, goto, finally and throw.
- These jump statements are used to move execution from one statement to another.
- The break statement terminates the loop (while, do-while, for, foreach statements) or switch statement and control is transferred to the next executing block of code (if any).

Example:

```
int x = 10;
do
{
    Console.WriteLine(x);
    x++; //Increment the value of x
}
while (x < 20);
```

Result: Will print up to 9 without break

continue Statement

- The continue statement will terminate the current iteration of a loop (while, do-while, for, for each statement) and return execution to the top of the loop.

Example:

```
for (int x = 0; x < 100; x++)
{
    if (10 == x)
        continue;

}
```

Result: Will print 1 to 100 without break at x=10

goto Statement

- The goto statement transfers the program execution to the labeled statement.
- The label identifier tells the compiler where statement execution should move to.
- Its is not good programming practice to use the goto label, but there may be some rare instances when it becomes useful.

goto Statement

- **Example**

```
class GoTo
{
    static void Main(string[] args)
    {

        int x = 10;
        if (x == 15)
        {
            goto Finish;
        }

    Finish:
        Console.WriteLine("End of search.");
        Console.ReadLine();
    }
```

checked Statement

- The checked statement causes all expressions in the block to be evaluated in a checked context.

Example:

```
class Checked
{
    static void Main(string[] args)
    {
        Check check = new Check();
        check.CheckMethod();


    }
    public class Check
    {
        public void CheckMethod()
        {
            int z = 0;
            int x = 10;
            int y = 10;
            try
            {
                z = checked((short)(x + y));
                Console.WriteLine(z);
                Console.ReadLine();
            }
            catch (System.OverflowException e)
            {
                Console.WriteLine(e.ToString());
            }
        }
    }
```

unchecked Statement

- The unchecked statement causes all expressions in the block to be evaluated in an unchecked context.

Example:

```
class Unchecked
{
    static void Main(string[] args)
    {
        int z;
        int x = 10;
        int y = 20;
        unchecked
        {
            z = x * y;
        }
        Console.WriteLine("Unchecked output Value:{0}",z);
    }
}
```

Constant & Readonly fields

- constant is a variable that contains a value that cannot be changed .
- The read-only keyword gives a bit more flexibility than const, allowing for situations in which you might want a field to be constant but also need to carry out some calculations to determine its initial value.
- The rule is that you can assign values to a read-only field inside a constructor, but not anywhere else.
- It's also possible for a read-only field to be an instance rather than a static field, having a different value for each instance of a class.

Q & A

- **Allow time for questions from participants.**

Test Your Understanding

- What are different types of data in C# 4.0?
- What is Boxing and Unboxing?
- What are different types operators in c#?
- What are control flow statements?

Data Types and Control Flow: Summary

Boxing and UnBoxing

- Converting a value-type into reference type is called boxing and the vice-versa is called Un-Boxing.
- Control flow statements –
- Control flow statements control the execution path of the program

Data Types and Control Flow : Summary

Here's a brief recap of this session:

Value Types and Reference Types –

Types that you define by using the struct keyword are value types and ones referred by Class keyword or reference types.

Operators

It is a symbol used to perform operations on one or more expressions.

Data Types and Control Flow : Source

Reference website:

www.msdn.microsoft.com

End of the Session

- **You have completed this Session of C# 4.0**

Arrays, Methods & Parameters in C#

About the Author

- Created By: Mohammad Salman
- Experience: 12 Years +
- Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Questions?

Reference

Try it Out

Informative Slide

Mandatory Slide

Welcome Break

Overview

Introduction:

- This session gives an overview of arrays, methods, parameters and method overloading.

Session Objectives

Objectives:

- After completing this session, you will be able to:
- Identify arrays
- Identify different array types
- Define Methods
- Define Parameters
- Describe Method Overloading

C# Arrays

- An Array is a data structure that contains several variables of the same type. Arrays are declared with a type.
- There are three types of array. They are,

C# Arrays

Single Dimensional:

```
int[] array1 = new int[5];
int[] array2 = new int[] { 1, 3, 5, 7, 9 };
int[] array3 = { 1, 2, 3, 4, 5, 6 };
```

Two / Multi Dimensional:

```
int[,] multiDimensionalArray1 = new int[2, 3];
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Jagged Array:

```
int[][] jaggedArray = new int[6][];
jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
```

Functions and Parametres

- In C#, arguments can be passed to parameters either by value or by reference. Passing by reference enables function members, methods, properties, indexers, operators, and constructors to change the value of the parameters and have that change persist in the calling environment. To pass a parameter by reference, use the ref or out keyword.

Functions and Parametres

```
class Program {

static void Main(string[] args) {

int arg;

// Passing by value. // The value of arg in Main is not changed.

arg = 4;

squareVal(arg);

Console.WriteLine(arg);

// Output: 4 // Passing by reference.  // The value of arg in Main is changed.

arg = 4;
```

Functions and Parametres

```
squareRef(ref  arg);
Console.WriteLine(arg);
// Output: 16
}
static void squareVal(int valParameter) {
valParameter *= valParameter;
}
// Passing by reference
static void squareRef(ref  int refParameter) {
 refParameter *= refParameter;
}
}
```