Visual C# 12

Eng. Mahmoud Ouf

Lecture 01

.Net 8

What is .Net?

.Net is a:

- Free
- Open Source
- Cross-Platform
- Development Platform

Development of .Net

What is .Net?

.Net was invented by Microsoft since 2002.

Through this long time it passes through 3 phases:

.Net Framework	(2002 – 2022)
.Net Core	(2016 – 2019)
.Net	(2020 – ……)

Development of .Net

What is .Net Framework?

.Net Framework was the first phase

It was launched 13/2/2002 with .Net Framework 1.0 visual studio 2002

and retired 9/8/2022 with .Net Framework 4.8.1 visual studio 2022

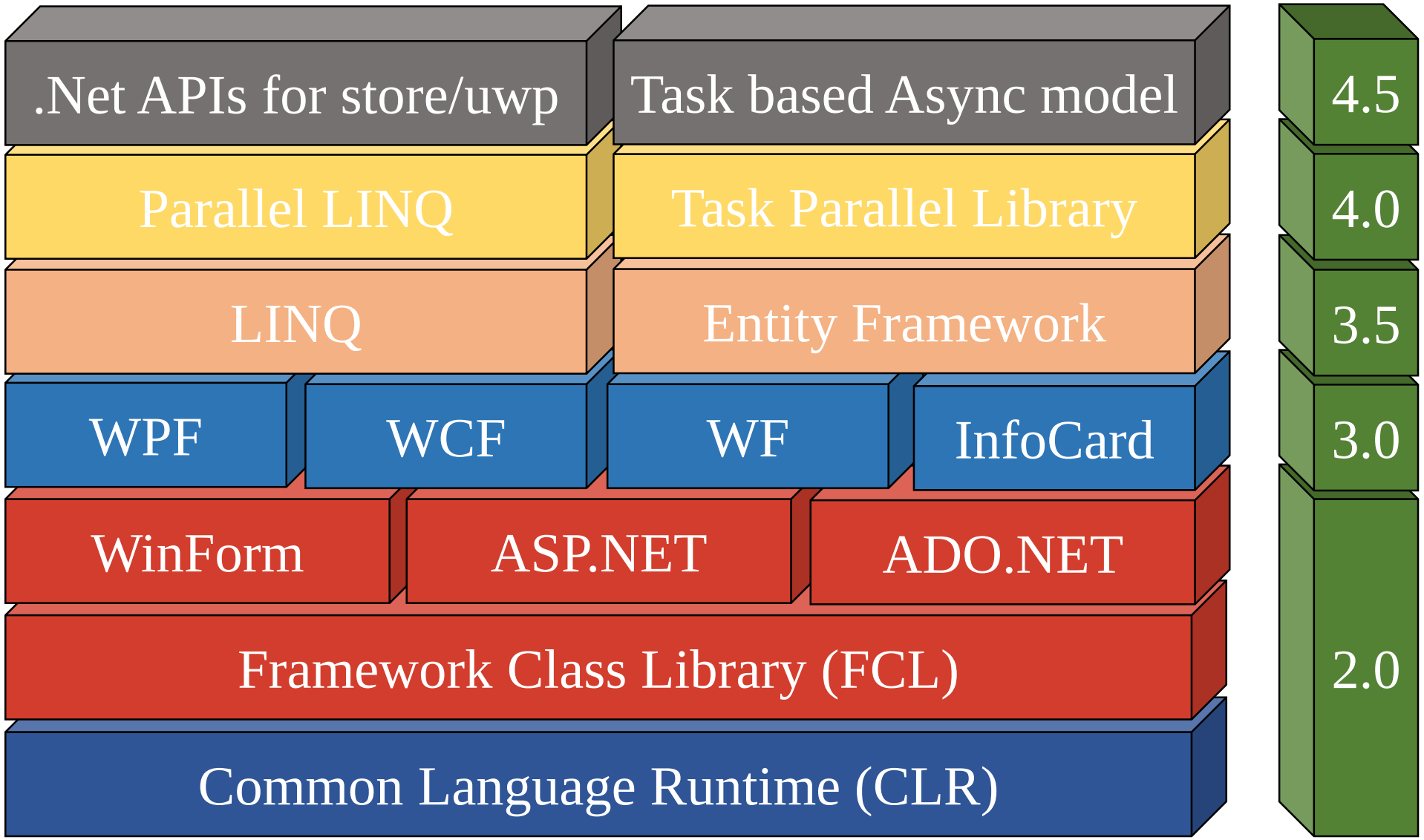
There was a great improvement in .Net Framework, but all the version are built on 2 important component

While the development was done by adding extra feature or another component.

The main Component are:

1. Common Language Runtime (CLR)
2. .Net Framework class Library (FCL)

Overview of .Net Framework



Development of .Net

What is .Net Core?

.Net Core was the second phase

It was launched 27/6/2016 with .Net Core 1.0 visual studio 2015

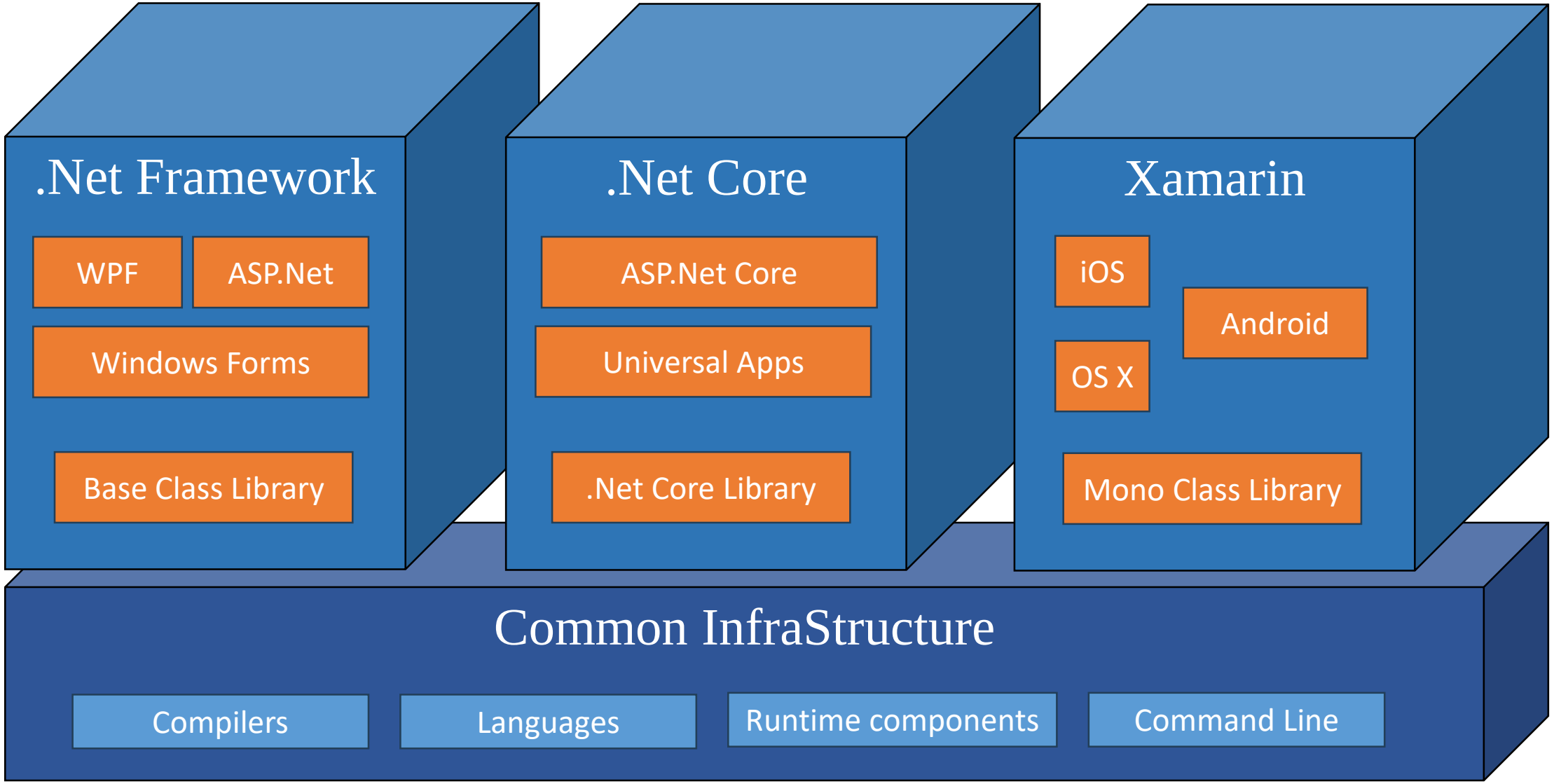
and retired 3/12/2019 with .Net Core 3.1 visual studio 2019

The need of .Net Core goes to the huge number of devices and operating system which lead to the requirement of building a consistent platform that can create different type of application.

The main Component are:

1. Core Common Language Runtime (Core CLR)
2. .Net Core class Library (Core FX)

Overview of .Net Core



Development of .Net

What is .Net?

.Net is the third phase

It was launched 10/11/2020 with .Net 5 visual studio 2019-U16.8

and last release 14/11/2023 with .Net 8 visual studio 2022-U17.8

As .Net Core build all type of application that .Net Framework do, it is not logic to keep and develop both platform.

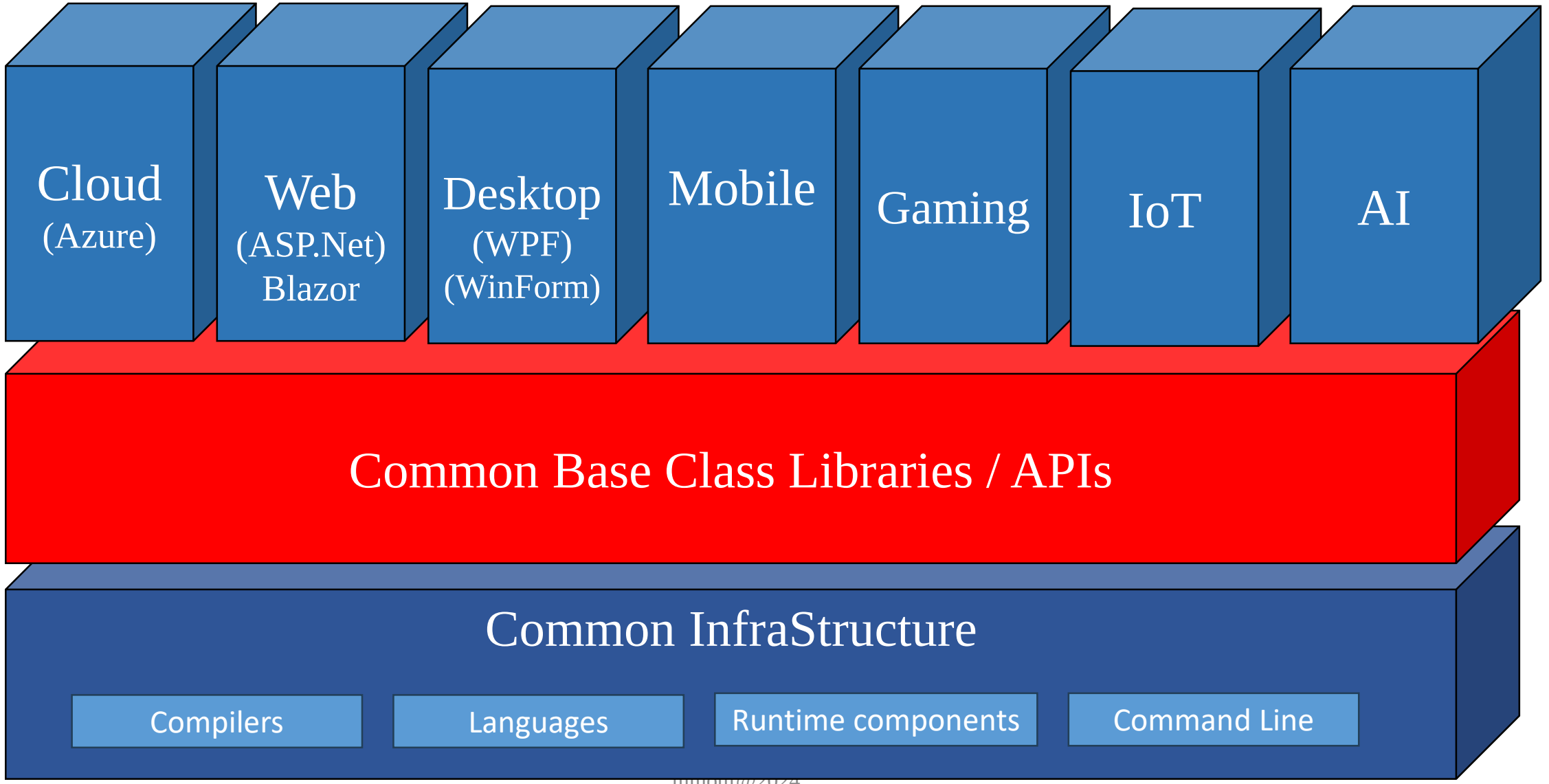
So, Microsoft decided to retire .Net framework and continue with .Net Core and renamed it .Net as it is the common development environment.

They couldn’t number it with 4 (it was used with .Net Framework)

The main Component are:

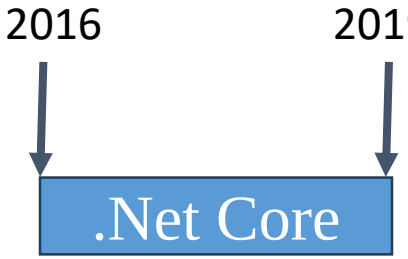
1. Core Common Language Runtime (Core CLR)
2. .Net Core class Library (Core FX)

Overview of .Net



Development of .Net

Date	.Net	C#
13/02/2002	1.0	1.0
09/04/2003	1.1	1.1
27/10/2005	2.0	2.0
06/11/2006	3.0	3.0
19/11/2007	3.5	3.5
12/04/2010	4.0	4.0
15/08/2012	4.5	5.0
20/07/2015	4.6	6.0
05/04/2017	4.7	7.0
18/04/2019	4.8	8.0
09/08/2022	4.8.1	8.1



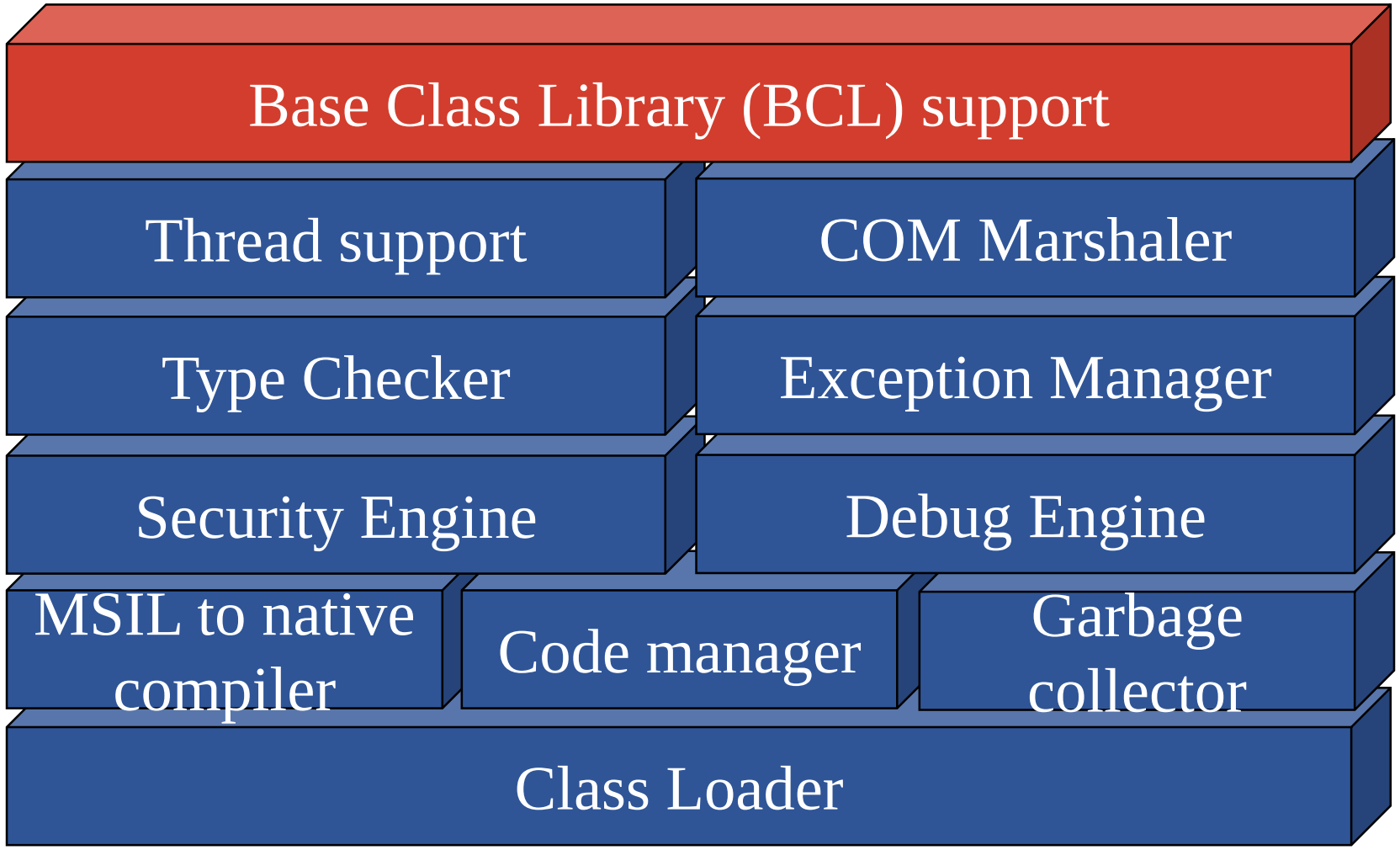
Date	ver	C#
27/06/2016	1.0	6.0
16/11/2016	1.1	6.0
14/08/2017	2.0	7.0
30/05/2018	2.1	7.0
04/12/2018	2.2	7.0
23/09/2019	3.0	8.0
03/12/2019	3.1	8.0

Date	ver	C#
10/11/2020	5	9
08/11/2021	6	10
08/11/2022	7	11

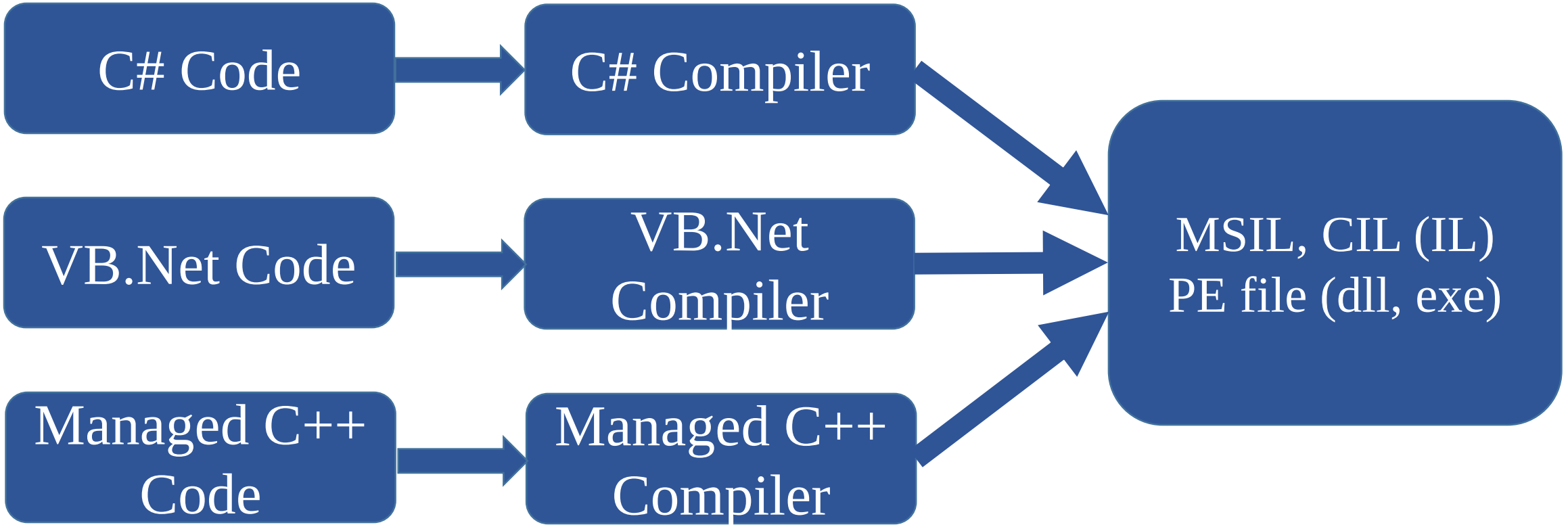
Introduction to .Net Framework

Framework version	Release date	Development tool
1.0	13/02/2002	Visual Studio .Net 2002
1.1	24/04/2003	Visual Studio .Net 2003
2.0	07/11/2005	Visual Studio .Net 2005
3.0	06/11/2006	Visual Studio .Net 2005
3.5	19/11/2007	Visual Studio .Net 2008
4.0	12/04/2010	Visual Studio .Net 2010
4.5	15/08/2012	Visual Studio .Net 2012
4.6	20/07/2015	Visual Studio .Net 2015

Common Language Runtime (CLR)



From source code to executable



Structure of C# program

- In C#, an application is a collection of one or more classes.
- The classes for a C# application can be written in more than one file and multiple classes can be put in one file. One class could be written in multiple files (partial class).

```
class HelloWorld
{
        public static void Main()
        {
                System.Console.WriteLine(“Hello, World”);
        }
}
```

Structure of C# program (cont.)

- The entry point to a C# application is the Main() method, which must be: contained in a class, begin with capital M, and public static.
- public: modifier tells us that the method is accessible by everyone.
- static: means that the method can be called without creating an instance of the class.
- The .Net Framework is made up of many namespaces, the most important of which is called **System**; it contains the classes that most application uses for interacting with the operating System.
- We can refer to objects in namespace by:
  1. prefixing them explicitly with the identifier of namespace
     System.Console.WriteLine("Hello, World");
  2. specifing a namespace by placing a using directive at the beginning of the application before the first class is defined
     using System;

Notes:

Top Level Statements (C# 9)

In C# 9, you can removed the class Program and the Main method

Ex:

```
// Display a simple message to the user.
Console.WriteLine("***** My First C# App *****");
Console.WriteLine("Hello World!");
Console.WriteLine();
// Wait for Enter key to be pressed before shutting down.
Console.ReadLine();
```

- Only 1 file in the application can use top level statement
- Top level statements is the entry point, so the program can't have another entry point
- Top level statement can't be enclosed in a namespace
- The top-level statements compile to a class named Program

Basic Input / Output operation

- **Output:**

Use the following 2 methods for output:
Console.Write()
Console.WriteLine()
The difference is that WriteLine() append a new line/carriage return to the end of the output.

To print a constant value:

```
Console.WriteLine(99);
Console.WriteLine("Hello, World");
```

To print a variable value:

```
int x = 99;
Console.WriteLine(x);
```

To print a combination from variable and constant value:

```
Console.WriteLine("The Sum of {0} and {1} is {2}", x, x, x+x);
```

Basic Input / Output operation

- **Output:**

String Interpolation (C# 6):

Using string interpolation in c#6, you can directly write your arguments instead of referring them with placeholders inside a string.

```
Console.WriteLine($“The Sum of {x} and {x} is {x+x}”);
```

Basic Input / Output operation

- **Output (Formatting Numerical Data):**

String Format Character

C or c	Used to format currency
D or d	Used to format decimal numbers, it may specify minimum number of digit
E or e	Used for exponential notation
F or f	Used for fixed point formatting, it may specify minimum number of digit
N or n	Used for basic numerical formatting (with commas)
X or x	Used for hexadecimal formatting

Basic Input / Output operation

- **Output (Formatting Numerical Data):**

Example:

```
static void Main()
{
        Console.WriteLine("The value 99999 in various formats:");
        Console.WriteLine("c format: {0:c}", 99999);
        Console.WriteLine("d9 format: {0:d9}", 99999);
        Console.WriteLine("f3 format: {0:f3}", 99999);
        Console.WriteLine("n format: {0:n}", 99999);
        Console.WriteLine("E format: {0:E}", 99999);
        Console.WriteLine("e format: {0:e}", 99999);
        Console.WriteLine("X format: {0:X}", 99999);
        Console.WriteLine("x format: {0:x}", 99999);
}
```

Basic Input / Output operation

- **Input:**

Use the following 2 methods for iutput:
Console.Read(), which read single character and return its ASCII
Console.ReadLine(), which read a string

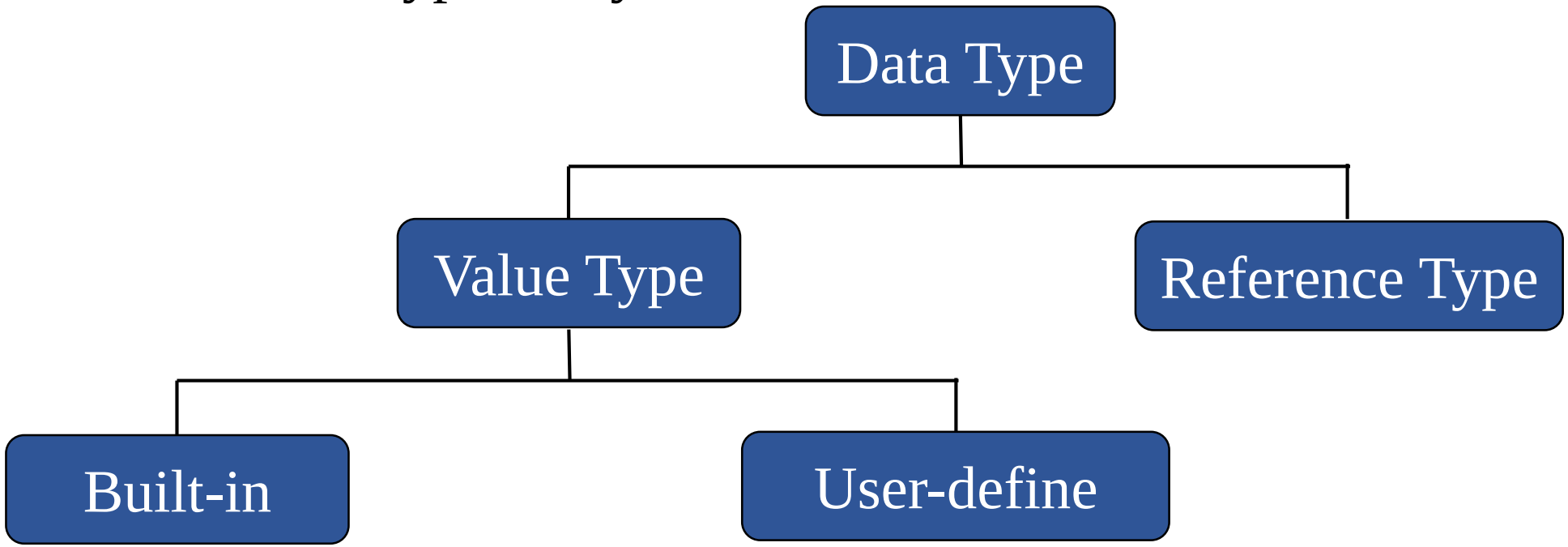
```
string input = Console.ReadLine();
```

To read an integer, use the Parse:

```
string s = Console.ReadLine();
int n = int.Parse(s);
```

Common Type System (CTS)

The CLR include Common Type System (CTS) that defines a set of built in data type that you can use.



Common Type System (CTS) (cont)

Value Type

The value type variables directly contain their data.

Reference Type

The reference type variable contains reference to their data. The data is stored in an object. It is possible for 2 reference type variables to reference the same object.

Note:

All data types are defined in System namespace.

All data types are derived from System.Object

Value types are derived from System.ValueType.

System.Object

System.ValueType

All value types directly contains data, and they can't be null

Common Type System (CTS) “Built-in”

Category	Class name (type in runtime)	C# data type	Range	Description
Integers	Byte	byte	0 to 255	8 bit unsigned integer(1byte)
	sByte	sbyte	-128 to 127	8 bit signed integer(1byte)
	Int16	short	-32,768 to 32,767	16 bit signed integer(2bytes)
	Int32	int	-2,147,483,648 to 2,147,483,647	32 bit signed integer(4bytes)
	Int64	long		64 bit signed integer(8bytes)
	UInt16	ushort	0 to 65,535	16 bit unsigned integer(2bytes)
	UInt32	uint	0 to 4,294,967,295	32 bit unsigned integer(4bytes)
	UInt64	ulong		64 bit unsigned integer(8byte)
Floating Points	Single	float	$1.5 * 10^{-45}$ to $3.4*10^{38}$	A single-precision (32-bit) floating-point number.
	Double	double	$5.0*10^{-324}$ to $1.7*10^{308}$	A double-precision (64-bit) floating-point number.
	Boolean	bool	true/false	
	Char	char		A Unicode (16-bit) character

Declaring a variable

Declaring local variable:

Data_type var_name;

In C#, you can't use uninitialized variable.

You can assign value to variable when declaring it.

Use ***default*** keyword to assign a default value to variable (C# 7.1)

Operator:

Arithmatic operator : + - * / %

Relational operator : < > <= >= == != is

Conditional operator : && || ?:

Increment/Decrement : ++ --

Assignment operator : = *= /= += -=

Logical operator : & | !

Notes:

- Starting from (C# 7), the default literal (Keyword) can be to give a default value for a variable.
  Ex: int x = default; //give x value of 0
- You can create a reference type to the primitive data type
  Ex: int x = new int(); //this will set the value of x to 0
  From (C# 9) an update in the new operator to use with primitive data type
  Ex: int x = new()
- C# 7, introduce the _ as a digit separator
  Ex: Console.WriteLine(123_456);
- C# 7, introduce a new literal for binary numbers, it starts with 0b_ followed by the binary number
  Console.WriteLine("Sixteen: {0}",0b_0001_0000);

Creating User-Defined Data Types:

Enumeration

Enumerators are useful when a variable can only have a specific set of values.

Defining : enum Color {Red, Green, Blue}

Using : Color ColorPalette = Color.Red; OR

Color ColorPalette = (Color) 0;

Console.WriteLine("{0}", ColorPalette);

Note :

The enumeration element are of type int and the first element has a value of 0, each successive element increase by 1. You can change the behavior as follow:

enum Color {Red = 102, Green, Blue}; //Green = 103, Blue = 104

enum Color {Red = 102, Green = 10, Blue = 97};

Creating User-Defined Data Types:

Enumeration

By default, the enum is stored in Int32 type, while you can change it to be any other type

Ex: enum Color : byte{Red, Green, Blue}

Creating User-Defined Data Types:

Structure

Defining :

```
struct Employee
{
        public          string firstName;
        public          int     age;
}
```

Using:

```
Employee    CompanyEmployee;
CompanyEmployee.firstName = “Aly”;
```

Creating User-Defined Data Types:

Note:

You can create a structure variable using the new Keyword

Ex: Employee emp = new Employee();

This will invoke the default constructor (assign 0 to all element)

C# 10, provides you with the option of creating a parametrized constructors

```
struct Employee
{
        public      string firstName;
        public      int    age;
        public Employee(string s, int a)
        {age = a;     firstName = s;}
}
```

Converting Data Types

Implicit :

We can convert implicitly within the same type for example (int to long). It couldn't fail, but, may loose precision, but not magnitude.

Ex: using System;

```
class Test
{
        public static void Main()
        {
                int     intValue = 123;
                long    longValue = intValue;
                Console.WriteLine("(long){0} = {1}", intValue, longValue);
        }
}
```

Converting Data Types

Explicitly :

We can convert variables explicitly by using the cast expression. Ex:

```
class Test
{
        public static void Main()
        {
                long   longValue = Int64.MaxValue;
                int    intValue = (int) longValue;
                Console.WriteLine("(int){0} = {1}", longValue, intValue);
        }
}
```

Control Statement:

A) Selection Statements

1) The if statement

```
if (Boolean-expression)
{
        statement to be executed;
}
else
{
        statement to be executed;
}
```

the if statement evaluates a boolean expression true or false. It is unlike C, NO implicit conversion from int to bool. Ex:

```
if(number % 2 == 0)
{       Console.WriteLine(“Even”); }
```

Control Statement:

A) Selection Statements

1) The if statement

But, if(number % 2) { } //error CS0029: Can't implicitly convert type 'int' to 'bool'.

```
if(number % 2 == 0)
{
        Console.WriteLine("Even");
}
else
{
        Console.WriteLine("Odd");
}
```

Control Statement:

A) Selection Statements

2) The switch Statement

Use the switch statements for multiple case blocks. Use break statement to ensure that no fall-through occurs the fall-through was C/C++ which is leaving the case without break in order to execute the following case also. This is not supported in C#.

This is sometimes a powerful feature, more often it is the cause of hard to find bug. The switch statement has the following:

1. test for equality
2. case label are constant
3. only one case constant
4. switch block can contain local variable
5. optional case : default

Control Statement:

A) Selection Statements

2) The switch Statement

The constant label must be: any integer type, char, enum or string
switch in C# can test for string
case null is permitted
The default must have a break
Note :
C# statements associated with one or more case labels can’t silently fall-through or continue to the next case label. So, use the break statement.
You can group several constants together, repeat the keyword case for each constant (without break).

Control Statement:

A) Selection Statements

2) The switch Statement

Example:

```
class Selections
{
        public static int Main()
        {
                Console.WriteLine("Welcome to world of .Net");
                Console.WriteLine("1 = C# \n2 = Managed C++ \n3 = VB.Net\n");
                Console.WriteLine("Please Select your implementation Language : ");
                string s = Console.ReadLine();
                int n = int.Parse(s);
                switch(n)
                {
                        case 1:
                                Console.WriteLine("Excellent choice");
                        break;
```

Control Statement:

A) Selection Statements

2) The switch Statement

```
                    case 2:
                        Console.WriteLine("Very Good choice");
                    break;
                    case 3:
                        Console.WriteLine("Not So Good choice");
                    break;
                    default:
                        Console.WriteLine("No choice at all");
                    break;
                }//end switch
        }//end main
}//end class
```

Control Statement:

B) Iteration Statement

1) The for loop statement

It has the following syntax:

```
for(initializer ; condition ; update)
{
        Statement to be repeated;
}
```

•You can declare more than one variable in the initialization of for loop But they must be of the same type.

•Also, you can declare more than one expression in the update statement

•The condition statement must be boolean

Control Statement:

B) Iteration Statement

2) The while statement

It repeatedly executes an embedded statement while a Boolean expression is true

It has the following syntax

```
while(condition)
{
        Statement to be repeated;
}
```

Control Statement:

B) Iteration Statement

3) The do..while statement

It is like the while, but it execute then check (i.e. the statement is evaluated at least once)

It has the following syntax:

```
do
{
        statement to be repeated;
}while(condition);
```