2.Introduction to C#. Net

2.1 introduction to C#

What is C#?

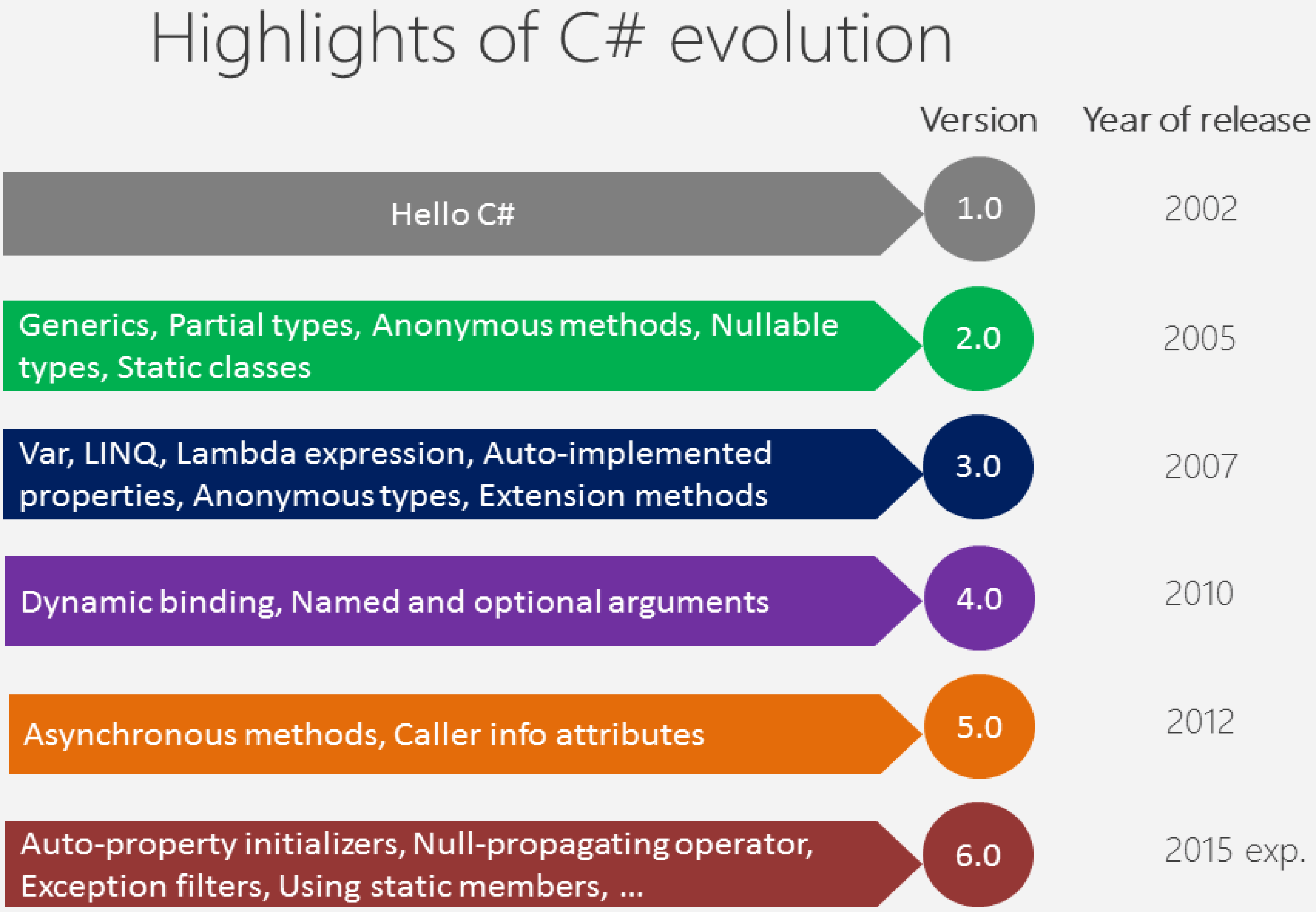
❑ C# (pronounced as ‘C Sharp’) is the language that has been designed from ground up with Internet in mind .

❑ It is modern language That combines the power of C++ with productivity of VB and elegance of Java.

❑ C# is a modern, type safe programming language, object oriented language that enables programmers to quickly and easily build solutions for the Microsoft .NET platform.

❑ Anders Hejlsberg introduced C# in .Net Framework.

2.3 History of C#? Design and Developed by **Anders Hejlsberg**.



Highlights of C# evolution

	Version	Year of release
Hello C#	1.0	2002
Generics, Partial types, Anonymous methods, Nullable types, Static classes	2.0	2005
Var, LINQ, Lambda expression, Auto-implemented properties, Anonymous types, Extension methods	3.0	2007
Dynamic binding, Named and optional arguments	4.0	2010
Asynchronous methods, Caller info attributes	5.0	2012
Auto-property initializers, Null-propagating operator, Exception filters, Using static members, …	6.0	2015 exp.

2.4 Unified Features of C# language

- ❑ A unified type system and simplifying the way that value and reference types are used by the language

- ❑ A component-based design established through features such as XML comments, attributes, properties, events and delegates.

- ❑ Practical developer headroom established through the unique capabilities of the C# language, including safe pointer manipulation, overflow checking, and more.

- ❑ Realistic language constructs, such as the foreach and using statements, which improve developer productivity.

2.5 Keywords in C#

Contains rich set of 76 reserved keywords. Keywords can be used as identifiers prefaced by an @.

Keywords	Keywords	Keywords	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
Out	Override	Params	Private
Protected	Public	Readonly	Ref

Keywords	Keywords	Keywords	Keywords
True	Try	Typeof	Uint
Ulong	Unchecked	Unsafe	Catch
Char	Checked	Class	Const
Using	Virtual	Void	While

Some special Keywords:

□ typeof is used to find out the managed type of type at run time.

Syntax: Type b1 = typeof(object);

Both 'is' and 'as' keywords are used for type casting in C#.

□ is operator is of boolean type whereas as operator is not of boolean type. The is operator returns true if the given object is of the same type whereas as operator returns the object when they are compatible with the given type.

Syntax: a1 is A;

□ as operator is used to perform conversion between compatible reference types or Nullable types. This operator returns the object when they are compatible with the given type and return null if the conversion is not possible instead of raising an exception.

Syntax: string s = myObjects[0] as string;

2.6 Data Types in C#

Most of the data type in C# are taken from C and C++. This tables lists data types, their description, and a sample example.

Type	Description	Example
object	The base type of all types	object obj = null;
string	String type - sequence of Unicode characters	string str = "Mahesh";

bool	Boolean type; a bool value is either true or false	bool val1 = true;
		bool val2 = false;

char	Character type; a char value is a Unicode character	char val = 'h';

Integer Data Types

TYPE	DESCRIPTION	MINIMUM VALUE	MAXIMUM VALUE
byte	Unsigned byte	0	255
sbyte	Signed byte	-128	127
short	Signed byte	-32 768	32 767
ushort	Unsigned byte	0	65 535
int	Signed byte	-2 147 483 648	2 147 483 647
uint	Unsigned byte	0	4 294 967 295
long	Signed long	-9x10^{8}	9x10^{8}
ulong	Unsigned long	0	1,8x10^{19}

Non-integer Data Types

All non-integer data types are signed.

TYPE	DESCRIPTION	PRECISION
float	Single Precision Number	7 digits
double	Double Precision Number	15 or 16 digits
decimal	Decimal Number	28 or 29 digits

Arithmetic Overflow

Overflow is an operation that occurs when a calculation produces a result that is greater in magnitude than that which a given register or storage location can store or represent.

2.6.2 Dynamic Data Type

The dynamic keyword brings exciting new features to C# 4.

Dynamic Type means that you can store any type of value in the dynamic data type variable because type checking for dynamic types of variables takes place at run-time.

```
dynamic dynInt = 100;
dynamic dynStr = "Hello";
```

2.7 Types in C#

C# supports two kinds of types: Value Types and Reference Types.

Types	Description	Types	Description
Value Types	Includes simple data types such as int, char, bool, enums	Value Types	Includes simple data types such as int, char, bool, enums
Reference Types	Includes object, class, interface, delegate, and array types	Reference Types	Includes object, class, interface, delegate, and array types

***Value Types-** Value type objects direct contain the actual data in a variables.
With value types, the variables each have their own copy of the data, and it is not possible for operations on one to affect the other.
int i = 10;

***Reference Types-** Reference type variables stores the reference of the actual data.
With reference types, it is possible for two variables to reference the same object, and thus possible for operations on one variable to affect the object referenced by the other variable.
MyClass cls1 = new MyClass();

Value Type	Reference Type
All simple types are value types.	All Complex Types are Ref. type except struct
Allocate memory at Compile Time.	Allocate memory at runtime.
Uses stack to store direct value.	Uses memory heap (block of memory) to store memory address rather than value.
Value type is more faster than Ref. type.	Ref. type is portable.
e.g. int, char, bool, enum,stuct. etc.	e.g. class, object (system.object), array, string, delegate etc.

2.8 What is Variable?

Variable is place holder for values in memory or simply variable is object of data Type.

How can we Declare and assign a value to the Variable:

- Declaration:

```
<type> <variable name>
int     a;
string  name;
```

- Assignment/Intialization:

```
A=10;
Name=“XYZ”;
```

2.9 var Type

var type is special data type to store varient type of data like an Object type. But Object is single culture and var is a multi culture type. We can store variant type of data into var as well as we can store multiple values of different type.

```
static void Main(string[] args)
      {
         var name = "Welcome";
         var a = 34;
         Console.WriteLine("name={0} a={1}",name,a);
         Console.ReadLine();
      }
```

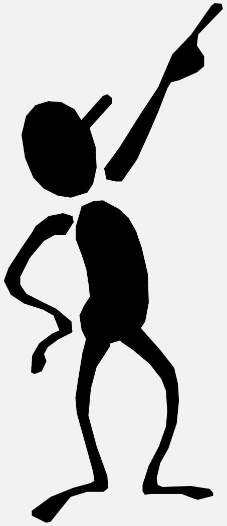
Using of C# var keyword in LINQ

```
static void Main(string[] args)
        {
            int[] array = { 1, 2, 4, 6, 8, 9, 11, 14, 15 };

            var a = from i in array where i % 2 == 0 select i;

            foreach(var _a in a)
            {
                Console.WriteLine("{0}\n",_a);
            }
            Console.Read();
        }
```

2.10 Skelton Of C# Program?

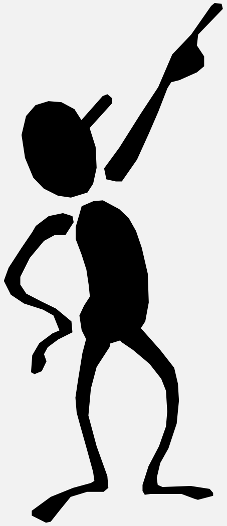
Console Based program

```
using  System;
using  System.Collections;
#region entry point
public  class  EntryClass
{
        public  static  void Main()
        {
          NewClass t = new NewClass();
        }
}
#endregion entry point
```



2.11 **Sample C# program-**Console Based program

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("HELLO WORLD");
            System.Console.ReadLine();
        }

    }
}
```



OUTPUT:

```
HELLO WORLD
```

2.12 Operators in C#

*Arithmetic Operators:

Operator	Function	Example
+	addition	11 + 2
-	subtraction	11 - 2
*	multiplication	11 * 2
/	division	11 / 2
%	remainder	11 % 2
++	Increment by 1	++expr1; expr2++;
--	Decrement by 1	--expr1; expr2--;

*Relational Operators

Operator	Function	Example
<	Less than	expr1 < expr2;
>	Greater than	expr1 > expr2;
<=	Less than or equal to	expr1 <= expr2;
>=	Greater than or equal to	expr1 >= expr2;
==	Equality	expr1 == expr2;
!=	Inequality	expr1 != expr2;

* Conditional Operators

Operator	Function	Example
!	Logical NOT	! expr1
||	Logical OR (short circuit)	expr1 || expr2;
&&	Logical AND (short circuit)	expr1 && expr2;
?:	Ternary	cond_expr ? expr1 : expr2;

The conditional operator takes the following general form:

expr

? execute_if_expr_is_true   : execute_if_expr_is_false;

2.13 Constants :

Classes and structs can declare constants as members.

Constants are values which are known at compile time and do not change. Constants are declared as a field, using the const keyword before the type of the field. Constants must be initialized as they are declared.
For example:

Multiple constants of the same type can be declared at the

same time, for example:

*How To Use Constant?

```
class Calendar2{
 static void Main(string[] args)
        {

             const int months = 12, weeks = 52, days = 365;

             const double daysPerWeek = days / weeks;

             const double daysPerMonth = days / months;


            System.Console.WriteLine(daysPerWeek);

             System.Console.WriteLine(daysPerMonth);

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

2.14 Type Conversion:

There are two types of type conversion *implicit* and *explicit*.

Implicit: Automatic compiler conversion where data loss is not an issue.

e.g. int iVal = 34;

long lVal = iVal;

Explicit: A conversin where data loss may happen and is recommended that the programmer writes additional processing

e.g. long lVal = 123456;

int iVal = (int) lVal;

*** Csharp Implicit Conversion:**

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 2;
            double y = 12.2;
            double z;
            z = x + y;     //x is automatically converted into a double
            Console.WriteLine(z);

            Console.Read();

        }
    }
}
```

*** Csharp Explicit Conversion:**

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
           double x = 2.1;
            int y = 12;

             int z = (int)x + y;      //Explicit conversion from double to int

            Console.WriteLine(z);

            Console.Read();
        }
    }
}
```

* Conversion by Convert Class:

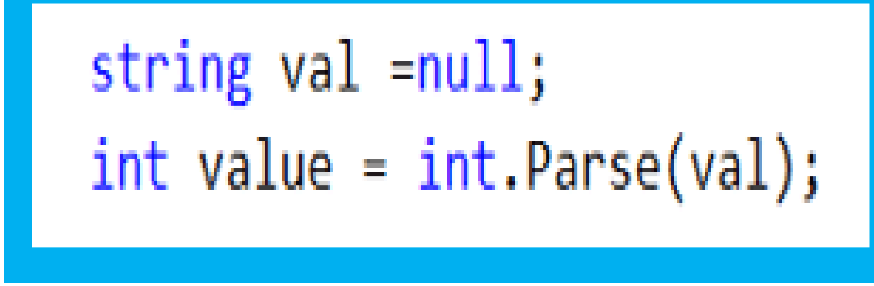
```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
           int y;
            y = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine(y);

            Console.Read();
        }
    }
}
```

* Conversion by Convert Class:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {

            double lVal = 123.45;

            int iVal = Convert.ToInt32(lVal);        //double is converted to int

            Console.WriteLine(iVal);
            Console.Read();
        }
    }
}
```

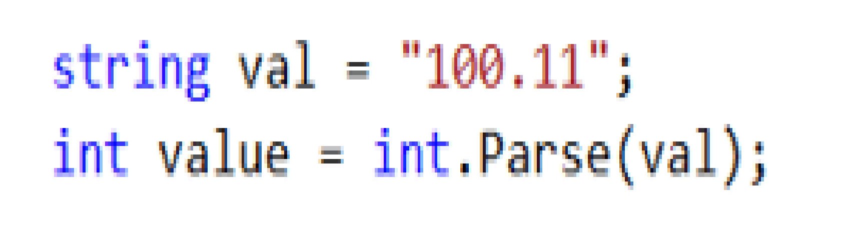
Int.Parse, Convert.ToInt32 and int.TryParse

	int.Parse	Convert.ToInt32	int.TryParse
	int.Parse(string s)	Convert.ToInt32(bool value)	int.TryParse (string s,out int result)
string str = "20";	20	20	20
string str = null;	ArgumentNullException	0	Failed to Convert but returns 0 and doesn't throw the exception
string str = “abc";	FormatException	FormatException	Failed to Convert but returns 0 and doesn't throw the exception
string str = “555557454554545454545";	OverflowException	OverflowException	Failed to Convert but returns 0 and doesn't throw the exception
bool b = false;	Not Supporting to convert	0	Failed to Convert but returns 0 and doesn't throw an Exception



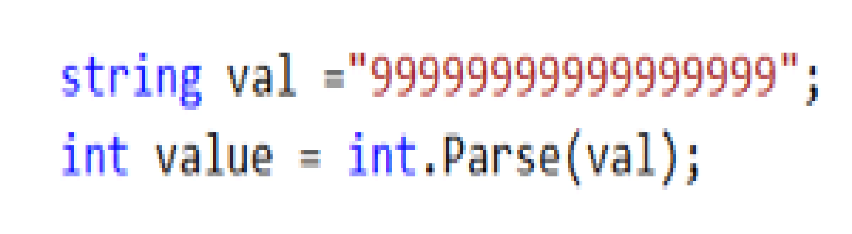
```
string val =null;
int value = int.Parse(val);
```

ArgumentNullException



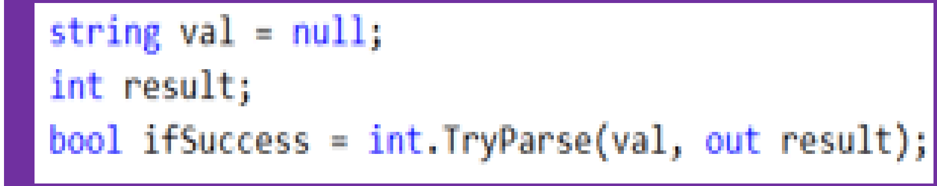
```
string val = "100.11";
int value = int.Parse(val);
```

FormatException



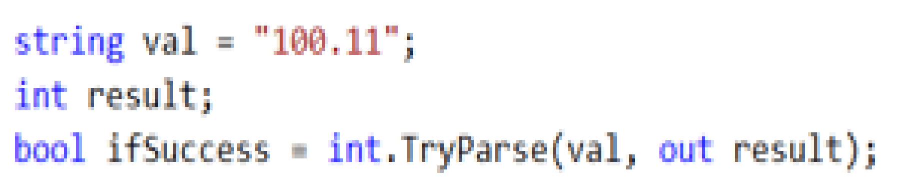
```
string val ="99999999999999999";
int value = int.Parse(val);
```

OverflowException



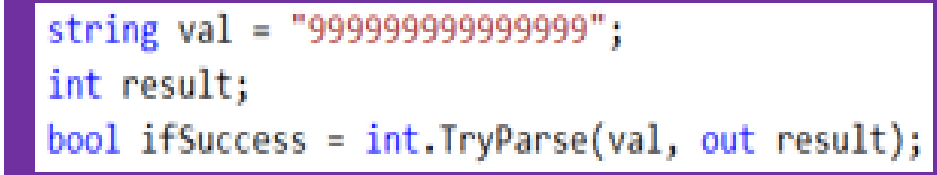
```
string val = null;
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0



```
string val = "100.11";
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0



```
string val = "999999999999999";
int result;
bool ifSuccess = int.TryParse(val, out result);
```

ifSuccess = false | result = 0

2.15 Boxing And UnBoxing

Object (System.Object) is the ultimate base class for all types.

Any type can be upcast to object.System.Object is such an important class that C# provides the object keyword as an alias for System.Object.

Boxing: To cast Value type to Object Type This process is known as “Boxing”.

```
static void Main(string[] args)
    {
        int i = 30;
        Object MyObj = i;
        Console.WriteLine(MyObj);
        Console.ReadLine();
    }
```

Unboxing: To cast Object type to Value Type This process is known as “Unboxing”.

```
static void Main(string[] args)
    {
        Object MyObj = 30;
        int i = (int)MyObj;
        Console.WriteLine(i);
       Console.ReadLine();
    }
```

Minutes of Chapter

- ❑ Introduction to C#.Net
- ❑ History of C#.Net
- ❑ Keywords in C#.Net
- ❑ Data Types in C#.Net
- ❑ Value Types And Reference Types
- ❑ Boxing And UnBoxing
- ❑ var Type
- ❑ Type Casting

Assignments for Chapter

Solve following problems with appropriate constructs by taking necessary inputs from user.

❑ WAP to Calculate Area of Circle Using Constants.

❑ WAP to Convert char value into ASCII values Using Type Casting and vise versa.

❑ WAP to Convert Dollar (double) into Rupees (int) and vice versa.

Some FAQ’s

1. **These are unified types in C#?**

   a. Classes and Struct.

   b. Value Types and Ref. Types.

   c. OOP and Non-OOP

   d. Heap and Stack.

2. **“Is” and “as” these keywords are used to?**

   a. perform type casting and check its compatibility.

   b. perform Garbage Collection.

   c. check equality.

   d. perform operator overloading.

3. **typeof is used to.**

a. find out managed type of an instance.

b. perform boxing and un-boxing.

c. find out the managed type of any type at run time

d. None of the above.

4. **type checking for ….types of variables takes place at run-time.**

a. value.

b. dynamic.

c. managed.

d. unmanaged.

5. …. is special data type to store variant type of data like an Object type but Object is single culture and … is a multi culture type.

a. dynamic type.

b. ref type.

c. value type.

d. var type.

6. type checking for ….types of variables takes place at run-time.

a. value.

b. dynamic.

c. managed.

d. unmanaged.

7. A Basic difference between int.Parse() and TryParse() is.

a. Parse returns int and TryParse return string.

b. Parse convert int to string and TryParse convert string to int.

c. Parse() method throws an exception if it cannot parse the value, whereas TryParse() method returns a bool indicating whether it succeeded.

d. None of the Above..

8. To cast Value type to Object Type This process is known as .

a. Un-boxing.

b. Type Casting.

c. Type Safety.

d. Boxing.