

Introduction to C#

Objectives

- Understand the features of C#
- Write Simple program in C# using Visual Studio.
- Create a DLL in C# and use it.



Introduction

- ▶ C# is a modern, object-oriented language that enables programmers to quickly build a wide range of applications for the new Microsoft .NET platform, which provides tools and services that fully exploit both computing and communications.
- ▶ C# supports the concepts of encapsulation, inheritance, and polymorphism.
- ▶ C# programming language can be used to develop different types of secured and robust applications.
 - Console Applications
 - Windows Applications
 - Web Applications
 - Distributed Applications
 - Web Service Applications

Features of C#

- ▶ Pointers are missing in C#.
- ▶ Unsafe operations such as direct memory manipulation are not allowed.
- ▶ Automatic Memory Management and Garbage Collection.
- ▶ Varying ranges of the primitive types like Integer, floats etc.
- ▶ Very powerful and simple for building interoperable, scalable, robust applications.
- ▶ Built in support to turn any component into a web service that can be invoked over the Internet from any application running on any platform.

Features of C#

OBJECT ORIENTED

- ▶ Supports Data Encapsulation, inheritance, polymorphism, interfaces
- ▶ C# introduces structures (structs) which enable the primitive types to become objects
 `int i = 1;`
 `string a = i.ToString(); //conversion (or) Boxing`

Features of C#

TYPE SAFE

- ▶ We cannot perform unsafe casts like convert double to a Boolean.
- ▶ Value types (primitive types) are initialized to zeroes and reference types (objects and classes) are initialized to null by the compiler automatically.
- ▶ Arrays are zero base indexed and are bound checked.
- ▶ Overflow of types can be checked.

Features of C#

INTEROPERABLE

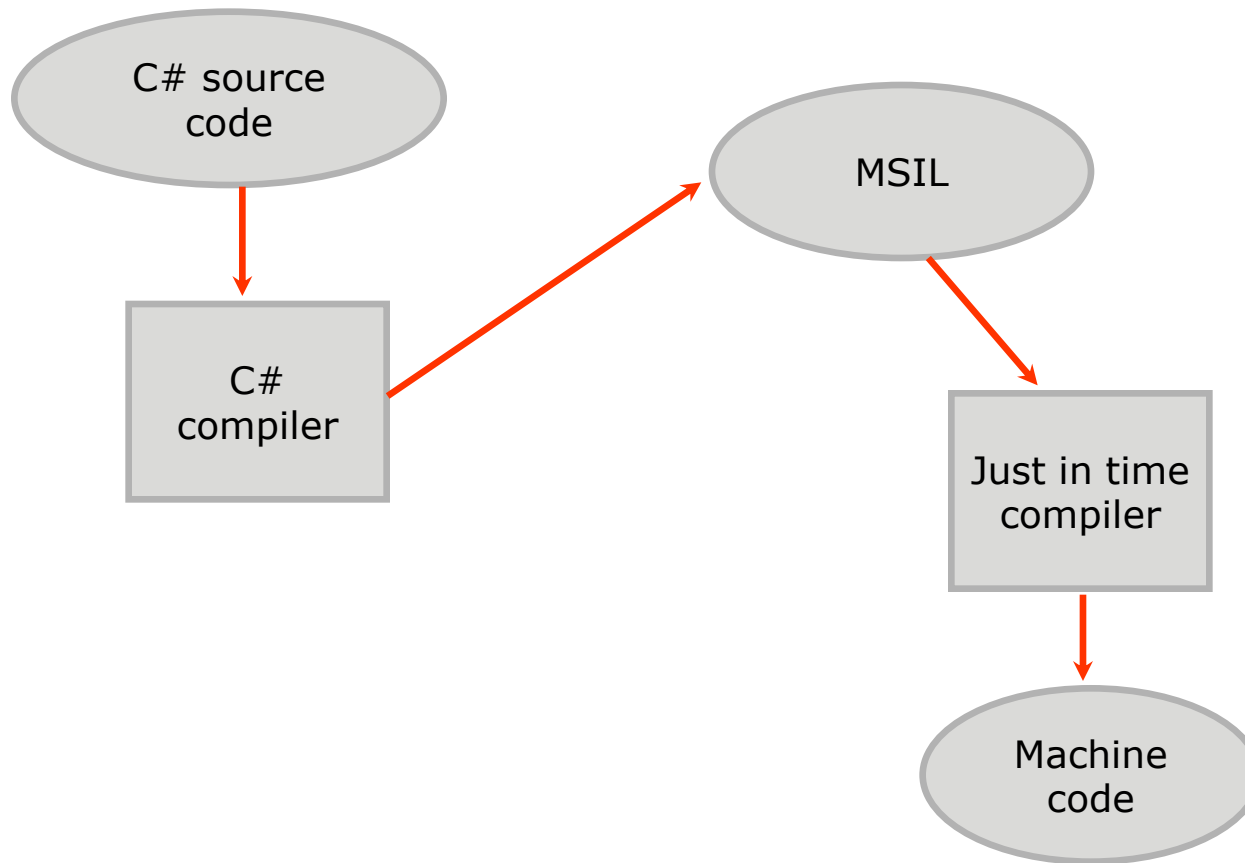
- ▶ It includes native support for the COM and windows based applications.
- ▶ C# allows the users to use pointers as unsafe code blocks to manipulate your old code.
- ▶ Components from VB NET and other managed code languages can directly be used in C#.

Features of C#

SCALABLE AND VERSIONABLE

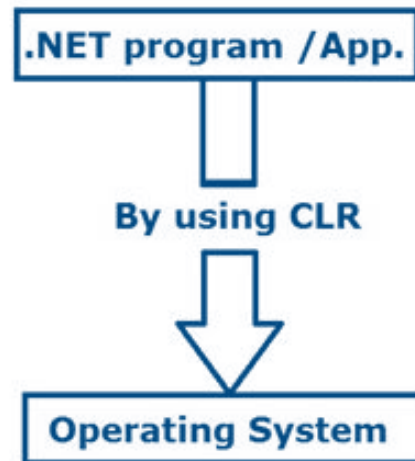
- ▶ .NET has introduced assemblies, which are self-describing by means of their manifest. Manifest establishes the assembly identity, version, culture and digital signature etc. Assemblies need not to be registered anywhere.
- ▶ No registering of dynamic linking library.
- ▶ C# support versioning in the language.

C# Program Execution



C# Program Compilation and Execution

- ▶ C# Program compilation steps are as follows.
 - Compiling Source Code into Managed Module.
 - Combining newly created managed module into the assembly / assemblies.
 - Loading the CLR.
 - Executing the assembly in & by CRL.



C# Program Compilation and Execution

- ▶ C# Programs can be compiled and executed using

- **Command Prompt**

- Save the file as **.cs** extension:

Hello.cs

- Compile the file using **csc** compiler:

csc Hello.cs

- Execute the file:

Hello.exe or Hello

- **Visual Studio IDE**

Dynamic Link Library (DLL)

- ▶ A Dynamic Link library (DLL) is a library that contains functions and codes that can be used by more than one program at a time.
- ▶ Both EXE and DLL files are executable program modules, but DLL cannot runs on its own. DLL needs to be taken in EXE to run it.
- ▶ DLL files cannot be executed directly.
- ▶ Once a DLL is created, it can be used in many applications.
- ▶ To use DLL
 - Add the reference/import the DLL file

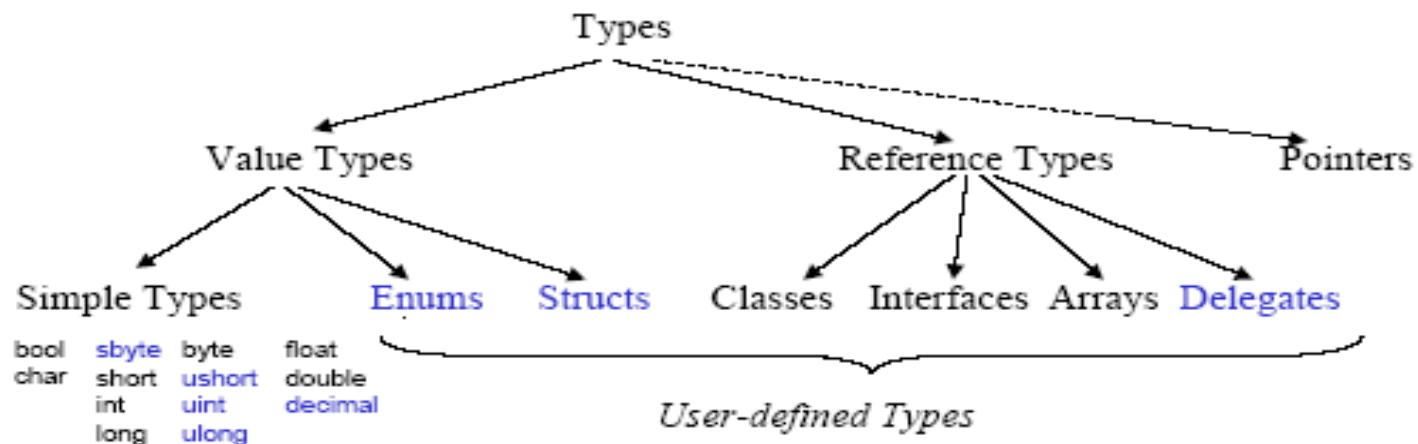
Test your memory

- ▶ What are the different features C# offers?
- ▶ How does the compilation take place for C# Program?
- ▶ Can you create DLL in C#? How?
- ▶ What is the difference between a .exe & a .dll?



Data Types in C#

Unified Type System



All types are compatible with *object*

- can be assigned to variables of type *object*
- all operations of type *object* are applicable to them

Data Types in C# - CLR Mapping

C# Type	CLR Type	C# Type	CLR Type
Bool	System.Boolean	UInt	System.UInt32
Byte	System.Byte	Long	System.Int64
SByte	System.SByte	Ulong	System.UInt64
Char	System.Char	Object	System.Object
Decimal	System.Decimal	Short	System.Int16
Double	System.Double	Ushort	System.UInt16
Float	System.Single	String	System.String
Int	System.Int32		

Data Types in C# - The System. Object type

- ▶ C# predefines a reference type named **object**.
- ▶ Every reference and value type is a kind of object. This means that any type we work with can be assigned to an instance of type object.
- ▶ For example:

```
object o;
```

```
o = 10;
```

```
o = "hello, object";
```

```
o = 3.14159;
```

```
o = new int[ 24 ];
```

```
o = false;
```


Data Types in C# - Value Types

- ▶ A variable of a value type always contains a value of that type.
- ▶ The assignment to a variable of a value type creates a copy of the assigned value
- ▶ Two Categories of value types:
 - Struct type: user-defined struct types, Numeric types, Integral types, Floating-point types, decimal, bool
 - Enumeration type

Data Types in C# - Reference Types

- ▶ Variables of reference types, referred to as objects, store references to the actual data.
- ▶ Assignment to a variable of a reference type creates a copy of the reference but not of the referenced object.
- ▶ Following are some of the reference types:
 - **class**
 - **interface**
 - **delegate**
- ▶ Following are built-in reference types:
 - **Object**
 - **String**

Data Types in C# - Value vs. Reference

Value Types	Reference Types
The variable contains the value directly	The variable contains a reference to the data (data is stored in separate memory area)
Allocated on stack	Allocated on heap using the new keyword
Assigned as copies	Assigned as references
Default behavior is pass by value	Passed by reference
== and != compare values	== and != compare the references, not the values
simple types, structs, enums	classes

Boxing and Unboxing

- ▶ Boxing and unboxing enable value types to be treated as objects. Value types, including both struct types and built-in types, such as `int`, can be converted to and from the type object.

- ▶ Boxing Example:

```
int nFunny = 2000;
```

```
object oFunny = nFunny;
```

- ▶ Now both the integer variable and the object variable exist on the stack, but the value of the object resides on the heap.

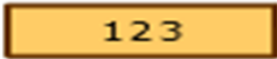
- ▶ **Unboxing Conversions**


```
int nFunny = 2000;
```

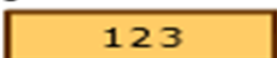
```
object oFunny = nFunny;
```

```
int nNotSoFunny = (int)oFunny
```

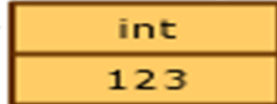
On the stack

i

`int i=123;`

o

`object o=i;`

j

`int j=(int) o;`

On the heap

(i boxed)


Arrays

- ▶ An array is a data structure that contains a number of variables called the elements of the array.
- ▶ C# arrays are zero indexed; that is, the array indexes start at zero.
- ▶ All of the array elements must be of the same type, which is called the element type of the array.
- ▶ Array elements can be of any type, including an array type.
- ▶ An array can be a single-dimensional array, a multidimensional array or a Jagged Array (Array of Arrays).
- ▶ Array types are reference type derived from the abstract base type **System.Array**.

Single Dimensional Array

- ▶ `// Declare a single-dimensional array of five integers`

```
int[] array1 = new int[5];
```

- ▶ `//An array that stores string elements`

```
string[] stringArray = new string[6];
```

- ▶ `// Declare and set array element values`

```
int[] array2 = new int[] { 1, 3, 5, 7, 9 };
```

- ▶ `// Alternative syntax`

```
int[] array3 = { 1, 2, 3, 4, 5, 6 };
```

Multi Dimensional Array

- ▶ `// Declare a two dimensional array`

```
int[,] multiDimensionalArray1 = new int[2, 3];
```

- ▶ `// Declare and set array element values`

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

Jagged Arrays (Array of Arrays)

- ▶ Jagged array is an array whose elements are arrays.
- ▶ The elements of a jagged array can be of different dimensions and sizes.
- ▶ A jagged array is sometimes called an "array of arrays."
- ▶ The following is a declaration of a single-dimensional array that has three elements, each of which is a single-dimensional array of integers:

```
int[][] jaggedArray = new int[3][];
```

```
jaggedArray[0] = new int[5];
```

```
jaggedArray[1] = new int[4];
```

```
jaggedArray[2] = new int[2];
```


Jagged Arrays (contd..)

- ▶ Each of the elements is a single-dimensional array of integers
- ▶ The first element is an array of 5 integers, the second is an array of 4 integers, and the third is an array of 2 integers.
- ▶ It is also possible to use initializers to fill the array elements with values, in which case you do not need the array size. For example:

```
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
```

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

```
jaggedArray[2] = new int[] { 11, 22 };
```

Arrays – More Declarations

► Single-Dimensional Array

- `int[] numbers = new int[5] {1, 2, 3, 4, 5};`
- `string[] names = new string[3] {"Matt", "Joanne", "Robert"};`
- `int[] numbers = {1, 2, 3, 4, 5};`
- `string[] names = {"Matt", "Joanne", "Robert"};`

► Multidimensional Array

- `int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };`
- `string[,] siblings = new string[2, 2] { {"Mike", "Amy"}, {"Mary", "Albert"} };`
- `int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };`
- `string[,] siblings = new string[,] { {"Mike", "Amy"}, {"Mary", "Ray"} };`
- `int[,] numbers = { {1, 2}, {3, 4}, {5, 6} };`
- `string[,] siblings = { {"Mike", "Amy"}, {"Mary", "Albert"} };`

Nullable Types

- ▶ Nullable types represent value-type variables that can be assigned the value of null.
- ▶ You cannot create a nullable type based on a reference type.
- ▶ Reference types already support the null value
- ▶ A nullable type can represent the normal range of values for its
- ▶ underlying value type, plus an additional null value
- ▶ A `Nullable<Int32>`, pronounced "Nullable of Int32," can be assigned any value from -2147483648 to 2147483647, or it can be assigned the null value.
- ▶ A `Nullable<bool>` can be assigned the values true or false, or **null**.

Why Nullable Types?

- ▶ The ability to assign null to numeric and Boolean types is particularly useful when dealing with databases and other data types containing elements that may not be assigned a value
- ▶ **Example:**
 - A Boolean field in a database can store the values **true** or **false**, or it may be undefined

Quick Recap

- ▶ The different features of C#.
- ▶ Create and execute C# program.
- ▶ How to Create and use a DLL in C#
- ▶ Data Types
- ▶ Arrays
- ▶ Nullable Types



Thank You

Atos, the Atos logo, Atos Syntel, and Unify are registered trademarks of the Atos group. © 2019 Atos.
Confidential information owned by Atos, to be used by the recipient only. This document, or any part of it, may not be reproduced, copied, circulated and/or distributed nor quoted without prior written approval from Atos.

Atos | Syntel