



C# Structure Programming Language Fundamentals

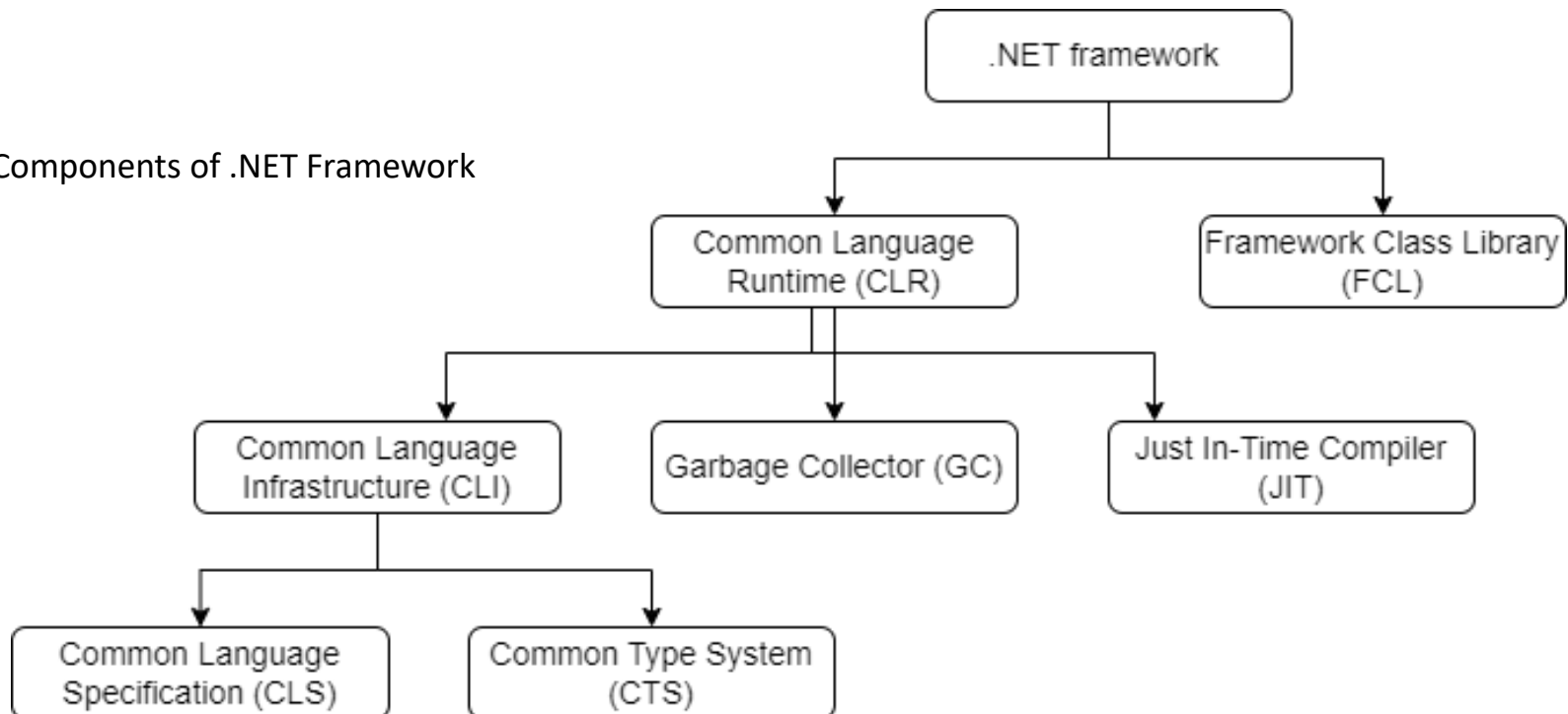
By: Võ Văn Hải

Email: vovanhai@ueh.edu.vn

Introduction to .NET Framework

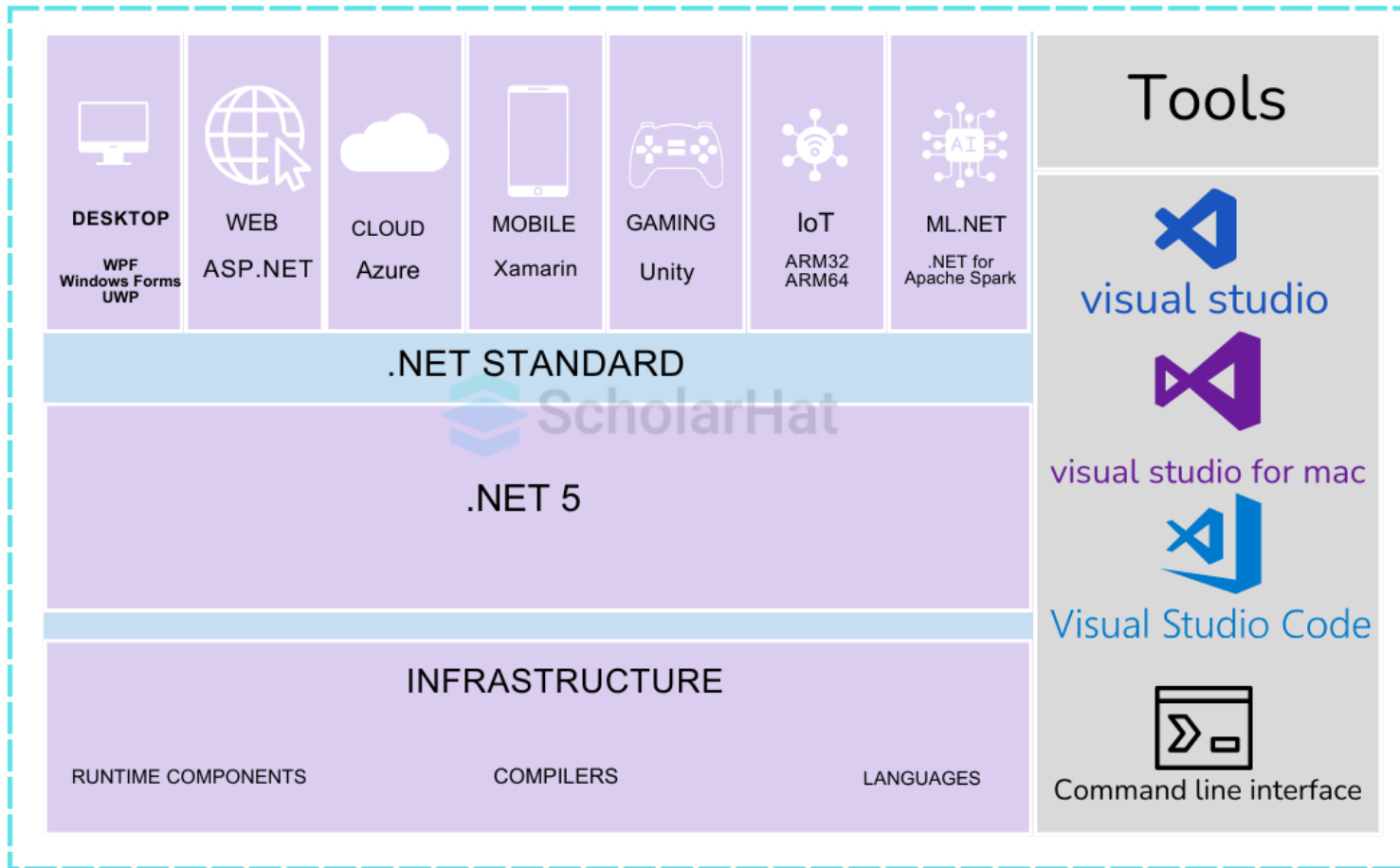
- ▶ The .NET Framework is a software development framework developed by Microsoft that provides a runtime environment and a set of libraries and tools for building and running applications on Windows operating systems.
- ▶ The framework includes a variety of programming languages, such as C#, F#, Visual Basic, etc., and supports a range of application types, including desktop, web, mobile, and gaming applications.

Main Components of .NET Framework



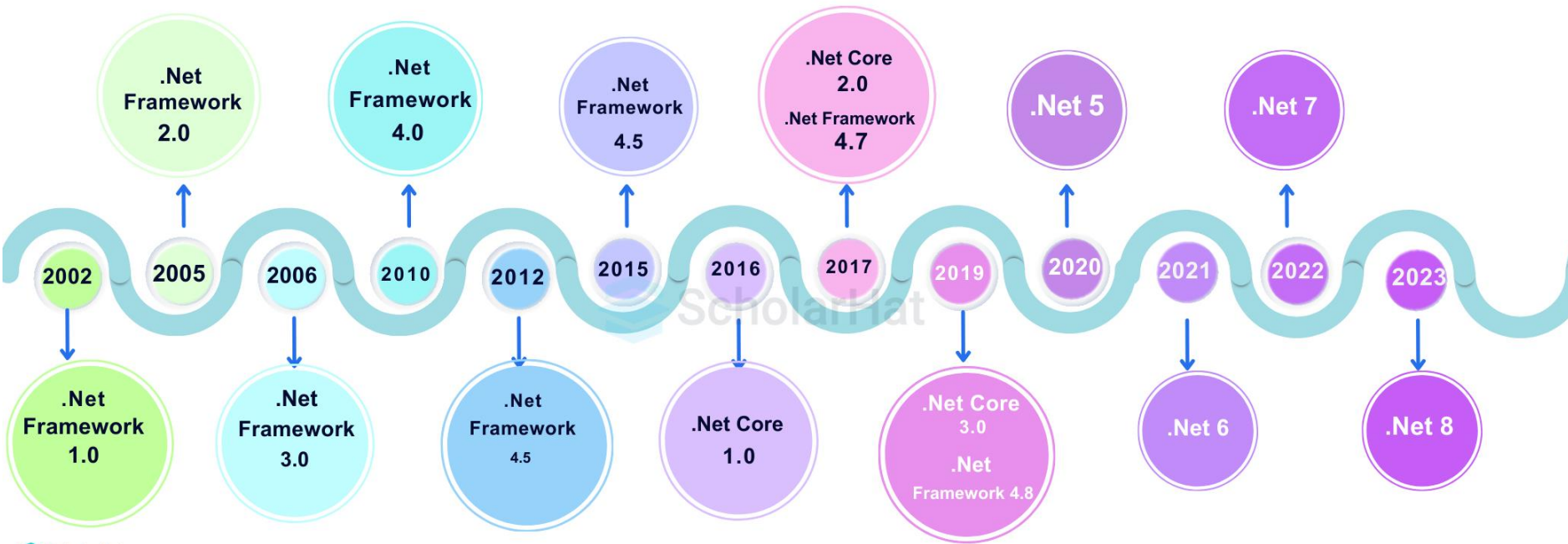
Introduction to .NET Framework

.NET Applications



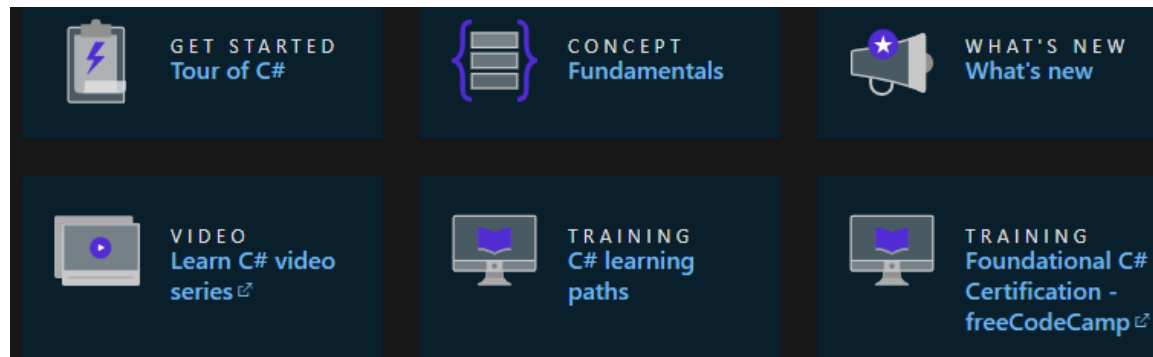
Introduction to .NET Framework

History of .NET



C# Introduction

- ▶ The C# programming language can be used to build software components and applications that run on a wide variety of operating systems (platforms)—including:
 - mobile devices,
 - game consoles,
 - web applications,
 - Internet of Things (IoT),
 - microservices, and
 - desktop applications.
- ▶ Resources from Microsoft:
 - <https://learn.microsoft.com/en-us/dotnet/csharp/>



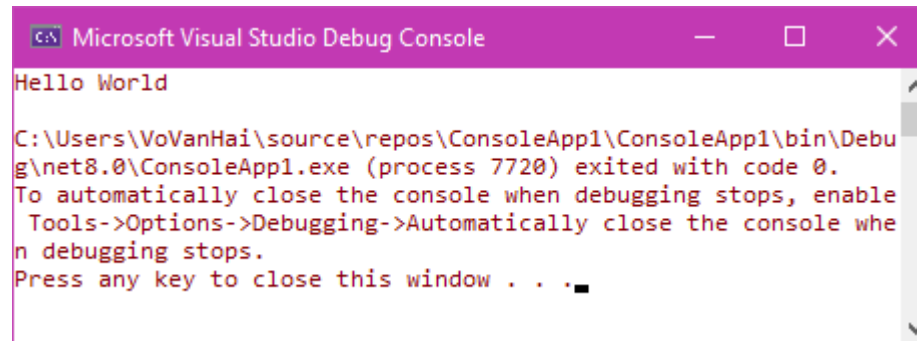
The HelloWorld example

► Morden style

```
1 Console.WriteLine("Hello World");
```

► Pre-C#8 style

```
1 using System;
2
3 class Program
4 {
5     private static void Main(string[] args)
6     {
7         Console.WriteLine("Hello World");
8     }
9 }
```



```
Microsoft Visual Studio Debug Console
Hello World
C:\Users\VoVanHai\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe (process 7720) exited with code 0.
To automatically close the console when debugging stops, enable
Tools->Options->Debugging->Automatically close the console whe
n debugging stops.
Press any key to close this window . . .
```

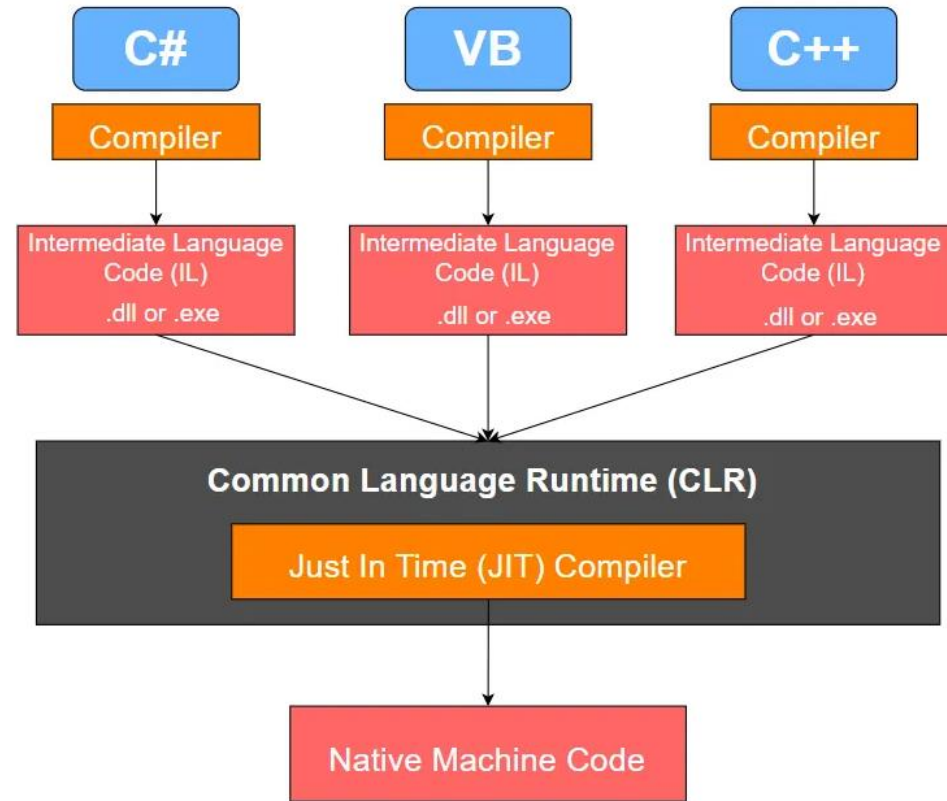


How can the computer execute this source code?

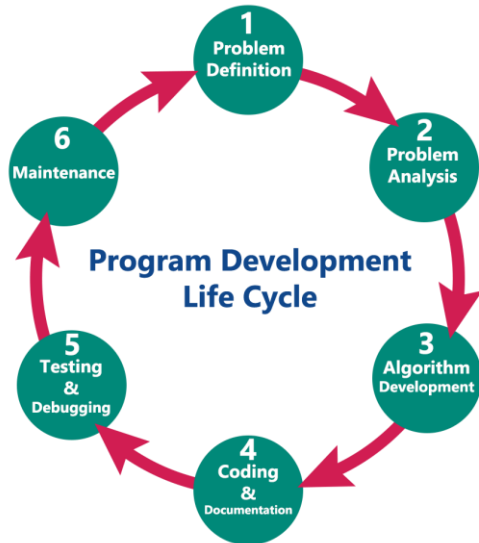
C# Compilation Process

Source Code

Managed Code



C# Compilation Process



Programming process

Create your first C# Application

- ▶ Install .Net Framework (version 8)
- ▶ Install Microsoft Visual Studio 2022 (Version > 17.10.x)
 - or Microsoft Visual Code with C# plugins

▶ Using Command-line

1. Make the folder and make it the current directory

2. Run command

```
dotnet new console
```

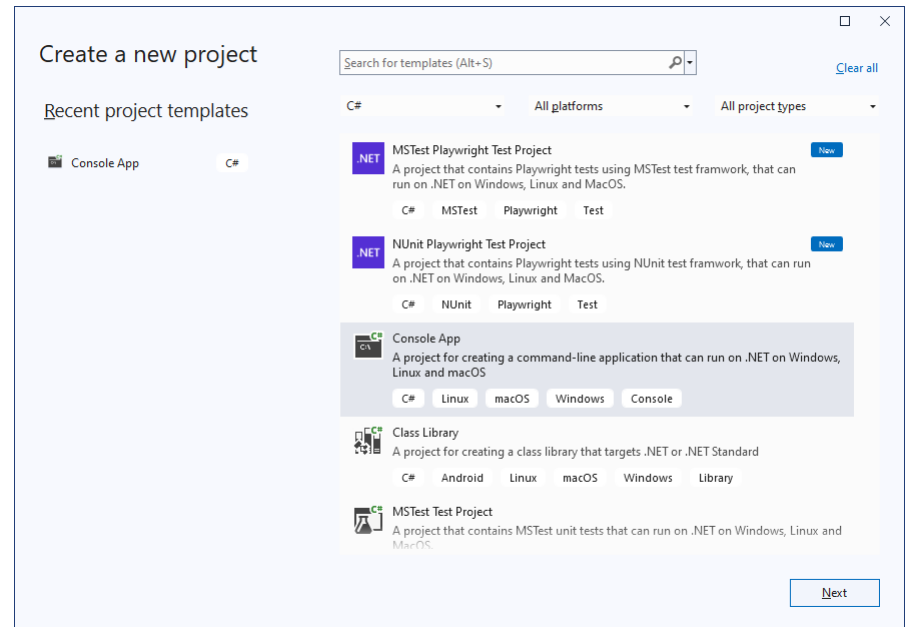
to create console application.

1. Run command

```
dotnet run
```

to run created application.

▶ Using Visual Studio



View demo

C# Fundamentals

[Language Specification](#)

Programming Languages

Introduction

- ▶ A programming language is a set of instructions and syntax used to create software programs. Some of the key features of programming languages include:
 - **Syntax:** The specific rules and structure used to write code in a programming language.
 - **Data Types:** The type of values that can be stored in a program, such as numbers, strings, and booleans.
 - **Variables:** Named memory locations that can store values.
 - **Operators:** Symbols used to perform operations on values, such as addition, subtraction, and comparison.
 - **Control Structures:** Statements used to control the flow of a program, such as if-else statements, loops, and function calls.
 - **Libraries and Frameworks:** Collections of pre-written code that can be used to perform common tasks and speed up development.
 - **Paradigms:** The programming style or philosophy used in the language, such as procedural, object-oriented, or functional.

Programming Languages

Basic Terminologies

- ▶ **Algorithm:** A step-by-step procedure for solving a problem or performing a task.
- ▶ **Variable:** A named storage location in memory that holds a value or data.
- ▶ **Data Type:** A classification that specifies what type of data a variable can hold, such as integer, string, or boolean.
- ▶ **Function:** A self-contained block of code that performs a specific task and can be called from other parts of the program.
- ▶ **Control Flow:** The order in which statements are executed in a program, including loops and conditional statements.
- ▶ **Syntax:** The set of rules that govern the structure and format of a programming language.
- ▶ **Comment:** A piece of text in a program that is ignored by the compiler or interpreter, used to add notes or explanations to the code.
- ▶ **Debugging:** The process of finding and fixing errors or bugs in a program.
- ▶ **IDE:** Integrated Development Environment, a software application that provides a comprehensive development environment for coding, debugging, and testing.
- ▶ **Operator:** A symbol or keyword that represents an action or operation to be performed on one or more values or variables, such as + (addition), – (subtraction), * (multiplication), and / (division).
- ▶ **Statement:** A single line or instruction in a program that performs a specific action or operation.

Programming Languages

General Structure of a C# Program

- ▶ C# programs consist of one or more files.
 - Each file contains zero or more namespaces.
 - A namespace contains types such as classes, structs, interfaces, enumerations, and delegates, or other namespaces.
- ▶ The program's **entry point** is the first line of program text in that file.
 - You can also create a **static** method named **Main** as the program's entry point

Reference: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line>

```
1  // A skeleton of a C# program
2  using System;
3  namespace YourNamespace
4  {
5      class YourClass {
6      }
7
8      struct YourStruct {
9      }
10
11     interface IYourInterface {
12     }
13
14     delegate int YourDelegate();
15
16     enum YourEnum {
17     }
18
19     namespace YourNestedNamespace {
20         struct YourStruct
21         {
22         }
23     }
24
25     class Program {
26         static void Main(string[] args) {
27             //Your program starts here...
28             Console.WriteLine("Hello world!");
29         }
30     }
31 }
```

Fundamentals

C Keywords and Identifiers

► Character set

A character set is a set of alphabets, letters and some special characters that are valid in C language.

- Alphabets

- Uppercase: A B C X Y Z
- Lowercase: a b c x y z

****C# accepts both lowercase and uppercase alphabets as variables and functions.**

- Digits

- 0 1 2 3 4 5 6 7 8 9

- Special Characters

,	<	>	.	_
()	;	\$:
%	[]	#	?
'	&	{	}	"
^	!	*	/	
-	\	~	+	

- White space Characters

- Blank space, newline, horizontal tab, carriage return and form feed.

Fundamentals

C# Keywords

- ▶ Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax, and they cannot be used as an identifier.
- ▶ As C# is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in C#.

abstract	add*(1)	alias*(2)	and*
args*	as	ascending*(3)	async*(5)
await*(5)	base	bool	break
by*(3)	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
descending*(3)	do	double	dynamic*(4)
else	enum	equals*(3)	event
explicit	extern	false	file*
finally	fixed	float	for
foreach	from*(3)	get*(1)	global*(2)
goto	group*(3)	if	implicit
in	init*(9)	int	interface
internal	into*(3)	is	join*(3)
let*(3)	lock	long	nameof*(6)
namespace	new	nint*(9)	not*
nonnull*(8)	null	nunit*(9)	object
on*(3)	operator	or*	orderby*(3)
out	override	params	partial*(2)
private	protected	public	readonly
record*	ref	remove*(1)	required*(11)
return	sbyte	scoped*	sealed
select*(3)	set*(1)	short	sizeof
stackalloc	static	string	struct
switch	this	throw	TRUE
try	typeof	uint	ulong
unchecked	unmanaged*(7.3)	unsafe	ushort

Fundamentals

C# Identifiers

- ▶ Identifier refers to name given to entities such as variables, functions, structures etc.
- ▶ Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program.
 - For example:

```
int money;  
double accountBalance;
```

- ▶ Here, *money* and *accountBalance* are identifiers.
- ▶ Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Fundamentals

Rules for naming identifiers

- ▶ A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- ▶ The first letter of an identifier should be either a letter or an underscore.
- ▶ You cannot use keywords like int, while etc. as identifiers.
- ▶ There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.
- ▶ You can choose any name as an identifier if you follow the above rule, however, give meaningful names to identifiers that make sense.
- ▶ Should read:



Naming convention

Variables, Constants and Literals

Variables

- ▶ In programming, a variable is a container (storage area) to hold data.
 - To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;  
char ch = 'a';  
ch = 'l';
```

Note: You should always try to give **meaningful names** to variables.

Variables, Constants and Literals

Literals

- ▶ Literals are data used for representing fixed values. They can be used directly in the code. For example: 1, 2.5, 'c' etc.

1. Integer

- An integer is a numeric literal (associated with numbers) without any fractional or exponential part. There are three types of integer literals in C programming:
 - decimal (base 10)
 - octal (base 8)
 - hexadecimal (base 16)

```
Decimal: 0, -9, 22 etc  
Octal: 021, 077, 033 etc  
Hexadecimal: 0x7f, 0x2a, 0x521 etc
```

2. Floating-point Literals

- A floating-point literal is a numeric literal that has either a fractional form or an exponent form. For example: -2.0, 0.000034, -0.22E-5

3. Characters

- A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc.

4. Escape Sequences

- Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc.

Variables, Constants and Literals

Escape Sequences

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

For example: \n is used for a newline. The backslash \ causes escape from the normal way the characters are handled by the compiler.

C Variables, Constants and Literals

Literals (cont.)

5. String Literals

- A string literal is a sequence of characters enclosed in double-quote marks. For example:

```
"good"           //string constant  
""              //null string constant  
"  "            //string constant of six white space  
"x"             //string constant having a single character.  
"Earth is round\n" //prints string with a newline
```

Variables, Constants and Literals

Constants

- ▶ If you want to define a variable whose value cannot be changed, you can use the **const** keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

- ▶ Here, PI is a symbolic constant; its value cannot be changed.

```
const double PI = 3.14;  
PI = 2.9; //Error
```

- ▶ You can also define many constants at the same time.

```
const int Months = 12, Weeks = 52, Days = 365;
```

- ▶ Or using expression to create a constant

```
const double DaysPerWeek = (double) Days / (double) Weeks;  
const double DaysPerMonth = (double) Days / (double) Months;
```

Data Types

Basic Types

- ▶ C# is a strongly typed language.
 - Every variable and constant has a type, as does every expression that evaluates to a value.
 - Every method declaration specifies a name, the type and kind (value, reference, or output) for each input parameter and for the return value.
- ▶ The .NET class library defines built-in numeric types and complex types that represent a wide variety of constructs.
 - These include the file system, network connections, collections and arrays of objects, and dates.
 - A typical C# program uses types from the class library and user-defined types that model the concepts that are specific to the program's problem domain.
- ▶ The information stored in a type can include the following items:
 - 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 interface(s) it implements.
 - The kinds of operations that are permitted.
 - The compiler uses type information to make sure all operations that are performed in your code are type safe.

Data Types

Specifying types in variable declarations

- ▶ When you declare a variable or constant in a program, you must either **specify its type** or use the **var** keyword to let the compiler infer the type.

```
// Declaration only:
```

```
float temperature;
```

```
string name;
```

```
MyClass myClass;
```

```
// Declaration with initializers (four examples):
```

```
char firstLetter = 'C';
```

```
var limit = 3;
```

```
int[] source = [0, 1, 2, 3, 4, 5];
```

```
var query = from item in source  
            where item <= limit  
            select item;
```

Data Types

Built-in types

- C# provides a standard set of built-in types.
 - These represent integers, floating point values, Boolean expressions, text characters, decimal values, and other types of data.
 - There are also built-in string and object types.

C# built-in **reference** types:

C# type keyword	.NET type
object	System.Object
string	System.String
dynamic	System.Object

C# built-in **value** types

C# type keyword	.NET type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
nint	System.IntPtr
nuint	System.UIntPtr
long	System.Int64
ulong	System.UInt64
short	System.Int16
ushort	System.UInt16

Reference: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/>

The background of the slide is a light blue gradient. It features a network of thin, light blue lines connecting various hexagonal shapes. Some hexagons are solid blue, while others are white with a blue outline. The hexagons are of different sizes and are scattered across the slide, creating a sense of a complex, interconnected system or network.

Basic Input/Output

C# Output

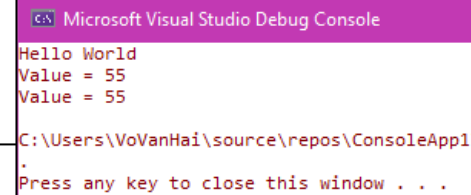
C# output

- In order to output something in C#, we can use

```
System.Console.WriteLine("You content with CR/LF");  
//OR  
System.Console.Write("You content without CR/LF");
```

- Here, *System* is a namespace, *Console* is a class within namespace *System* and *WriteLine* and *Write* are methods of class *Console*.

```
1  using System; //can be omitted  
2  namespace Sample {  
3      class Test {  
4          public static void Main(string[] args) {  
5              int val = 55;  
6              Console.WriteLine("Hello " + "World");  
7  
8              Console.WriteLine("Value = " + val); //concatenated string  
9  
10             Console.WriteLine("Value = {0}", val); //Formatted String  
11         }  
12     }  
13  
    int a = 5, b = 3, c = 4;  
    Console.WriteLine("{0} + {1} = {2}", a, b, c);
```



Microsoft Visual Studio Debug Console

```
Hello World  
Value = 55  
Value = 55  
C:\Users\VoVanHai\source\repos\ConsoleApp1  
"  
Press any key to close this window . . .
```

C# Output

String interpolation

- ▶ The \$ character identifies a string literal as an interpolated string.
- ▶ An interpolated string is a string literal that might contain interpolation expressions. When an interpolated string is resolved to a result string, the compiler replaces items with interpolation expressions by the string representations of the expression results.
- ▶ String interpolation provides a more readable, convenient syntax to format strings

```
public static void Main(string[] args){  
    var name = "Mark";  
    var date = DateTime.Now;  
  
    // Composite formatting:  
    Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.",  
        name, date.DayOfWeek, date);  
    // String interpolation:  
    Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, " +  
        $"it's {date:HH:mm} now.");  
    // Both calls produce the same output that is similar to:  
    // Hello, Mark! Today is Wednesday, it's 19:40 now.  
}
```

C# Input

- ▶ In C#, the simplest method to get input from the user is by using the `ReadLine()` method of the `Console` class.
 - However, `Read()` and `ReadKey()` are also available for getting input from the user. They are also included in `Console` class.

```
1  namespace Sample {  
2      class Test {  
3          public static void Main(string[] args) {  
4              string testString;  
5              Console.Write("Enter a string - ");  
6              testString = Console.ReadLine();  
7              Console.WriteLine("You entered '{0}'", testString);  
8  
9              int userInput;  
10  
11             Console.WriteLine("Press any key to continue...");  
12             Console.ReadKey();  
13             Console.WriteLine();  
14  
15             Console.Write("Input using Read() - ");  
16             userInput = Console.Read();  
17             Console.WriteLine("Ascii Value = {0}", userInput);  
18         }  
19     }  
20 }
```

- ***ReadLine()***: reads the next line of input from the standard input stream. It returns the same string.
- ***Read()***: reads the next character from the standard input stream. It returns the ascii value of the character.
- ***ReadKey()***: obtains the next key pressed by user. This method is usually used to hold the screen until user press a key.

C# Input

- ▶ The difference between `ReadLine()`, `Read()` and `ReadKey()` method is:
 - `ReadLine()`: reads the next line of input from the standard input stream. It returns the same string.
 - `Read()`: reads the next character from the standard input stream. It returns the ascii value of the character.
 - `ReadKey()`: obtains the next key pressed by user. This method is usually used to hold the screen until user press a key.

```
public static void Main(string[] args){  
    int userInput;  
  
    Console.WriteLine("Press any key to continue...");  
    Console.ReadKey();  
    Console.WriteLine();  
  
    Console.Write("Input using Read() - ");  
    userInput = Console.Read();  
    Console.WriteLine("Ascii Value = {0}", userInput);  
}
```

C# Input

Reading other types values

- ▶ Since the `ReadLine()` method receives the input as string, it needs to be converted into the specific type.
 - One simple approach for converting our input is using the methods of **Convert** class.

```
public static void Main(string[] args){
    Console.Write("Enter integer value: ");
    String userInput = Console.ReadLine();
    /* Converts to integer type */
    int intVal = Convert.ToInt32(userInput);
    Console.WriteLine("You entered {0}", intVal);

    Console.Write("Enter double value: ");
    userInput = Console.ReadLine();
    /* Converts to double type */
    double doubleVal = Convert.ToDouble(userInput);
    Console.WriteLine("You entered {0}", doubleVal);
}
```

- ▶ Other methods of `Convert` class can be found here:
<https://learn.microsoft.com/en-us/dotnet/api/system.convert?view=net-8.0>

Fundamentals

C# Comments

- ▶ Comments are used in a program to help us understand a piece of code. They are human readable words intended to make the code readable. Comments are completely ignored by the compiler.
- ▶ In C#, there are 3 types of comments:
 - Single Line Comments (`//`)
 - Multi Line Comments (`/* */`)
 - XML Comments (`///`)
- ▶ Comments are used to explain parts of code, but they should not be overused.
 - Instead, comments should be used in the program to explain complex algorithms and techniques.
 - Comments should be short and to the point instead of a long description.
 - As a rule of thumb, it is better to explain why instead of how, using comments.

```
/// <summary>
/// XML style method
/// </summary>
static void sampleMethod(){
    //Single line comment
    /*
     * multi-lines comment
     */
}
```

Fundamentals

Debugging

- ▶ Debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code.
- ▶ Debugging is a means of diagnosing and correcting the root causes of bugs (errors) that have already been detected.
- ▶ To debug a program:
 1. Start with the problem
 2. Isolate the source of the problem
 3. Fix it
- ▶ Debugging tools (*Debuggers*) help identify coding errors at various development stages.
- ▶ In VS, do the following steps for debugging
 1. Start the debugger and hit breakpoints.
 2. Learn commands to step through code in the debugger
 3. Inspect variables in data tips and debugger windows
 4. Examine the call stack



Exercises

Write programs that:

1. to Add / Sum Two Numbers.
2. to Swap Values of Two Variables.
3. to Multiply two Floating Point Numbers
4. to convert feet to meter
5. to convert Celsius to Fahrenheit and vice versa
6. to find the Size of data types
7. to Print ASCII Value (tip: read character, print number of this char)
8. to Calculate Area of Circle
9. to Calculate Area of Square
10. to convert days to years, weeks and days

Submit your exercise to Git-Hub in the folder named *Exercises-1*

The background of the slide is a light blue gradient. It features a network of thin, light blue lines connecting various hexagonal shapes. Some hexagons are solid blue, while others are white with a blue outline. The hexagons are of different sizes and are scattered across the slide, creating a sense of a complex, interconnected system or network.

Thanks for your listening!