C#
Programming

Introduction to Programming Using C#

Course Content

- Introduction to programming
- Variables and Data Types
- Operators and Control Statements
- Arrays
- Methods
- Structures and Enum

Introduction to Programming

- What is Program & Programming?
- ***Computer Program*** is a collection of instructions that can be executed by a computer to perform a specific task
- ***Computer Programming*** is the process of *designing* and *building* an executable computer program to accomplish a specific computing result or to perform a specific task
- ***Program Component***
  - □ *Data* (data structure)
  - □ *Instructions* (Algorithm s)

Classification of Programming Languages

- ***Low-level*** Programming Languages
  - Near to Hardware
- ***High-level*** Programming Languages
  - Near to Human Language

Low-Level Programming language

- is a programming language with little (Assembly) or zero (Machine Code) abstraction from the details of the computer.
- use the specific instruction set of a processor or little higher. The instruction set for each processor is defined by the manufacturer
- Ex :Machine code
- Ex: Assembly

Machine Code

- In computer programming, machine code is any *low-level* programming language, consisting of machine language instructions
- Ex: A function in hexadecimal representation of 32-bit x86 machine code to calculate the nth Fibonacci number:
- 0 1 1 2 3 5 8

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD989
C14AEBF1 5BC3
```

Assembly Language

- *Second-generation* languages that’s designed to communicate instructions with specific computer hardware
- Assembly Code translated into machine Code using Assembler
- Ex: The same Fibonacci Sequence calculator as previous, but in x86-64 assembly language using AT&T syntax

```
_fib:
        movl $1, %eax
        xorl %ebx, %ebx
.fib_loop:
        cmpl $1, %edi
        jbe  .fib_done
        movl %eax, %ecx
        addl %ebx, %eax
        movl %ecx, %ebx
        subl $1, %edi
        jmp  .fib_loop
.fib_done:
        ret
```

High level Programming language

- It is a programming language with strong abstraction from the details of the computer.
- Use English like statements
- Ex: The same Fibonacci number calculator as previous using C programming Language

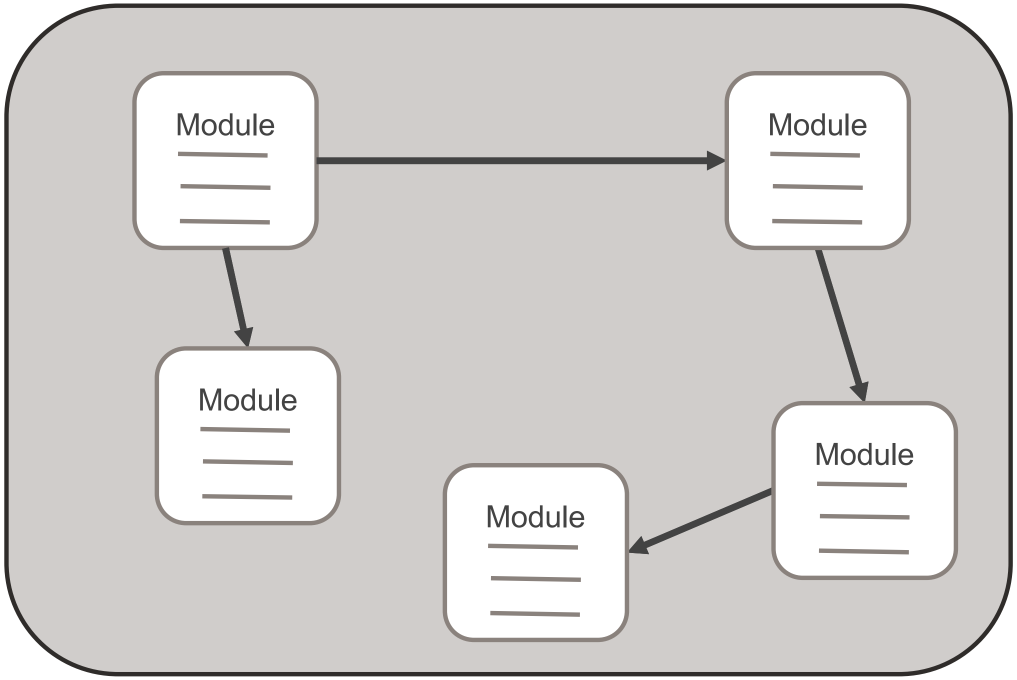
```
int fib(int n)
{
    if (!n)
         return 0;
    else if (n <= 2)
         return 1;
    else {
            int a, c;
         for (a = c = 1; ; --n)
         {
              c += a;
              if (n <= 2) return c;
              a = c - a;
         }
    }
}
```

Programming Languages Paradigms

- High Level languages Could be classified (Methodology-wise) into
  - **Structured or Modular Languages**
    - Program divided into smaller tasks called module to be called when needed
    - ex: C , Fortran , Basic
  - **Object Oriented Languages**
    - Program consist of Objects
    - ex: C++, Java, C#

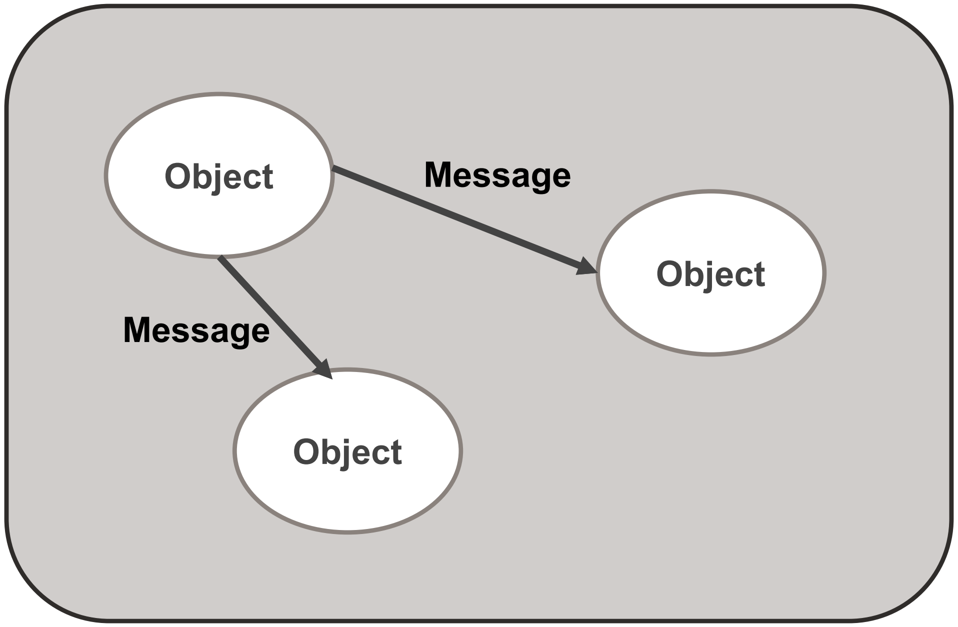
Modular Programming

- Ex: C , Fortran ,basic



Object Oriented Programming

- Ex: C++ ,Java ,C#

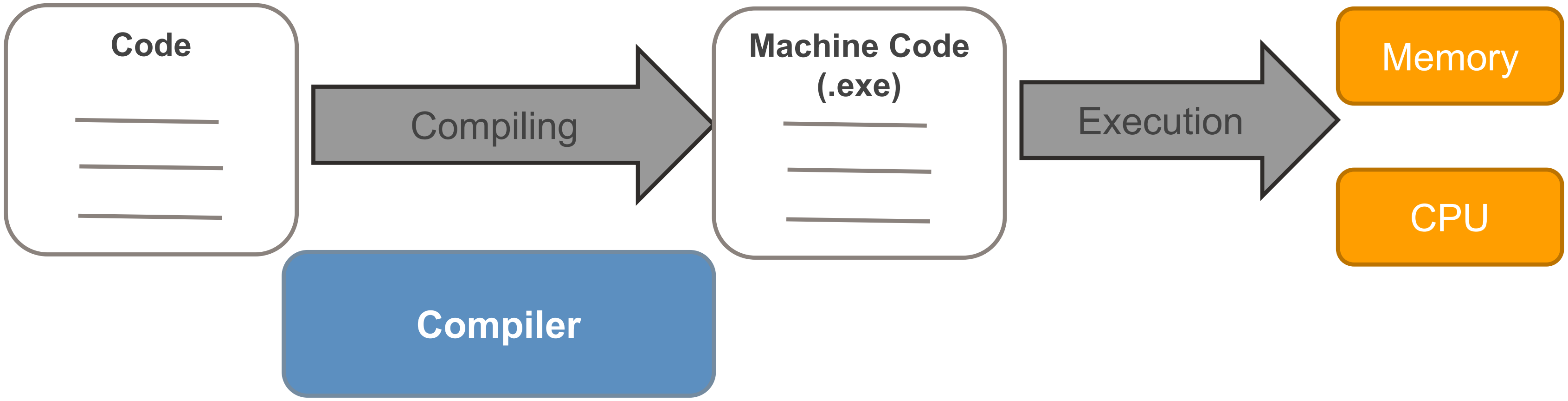


Program Life Cycle

- Start → file with code
- End → Run the program
- This process achieved by one of the following ways
  - Using **Compiler**
  - Using **Interpreter**
  - Hypered Mix

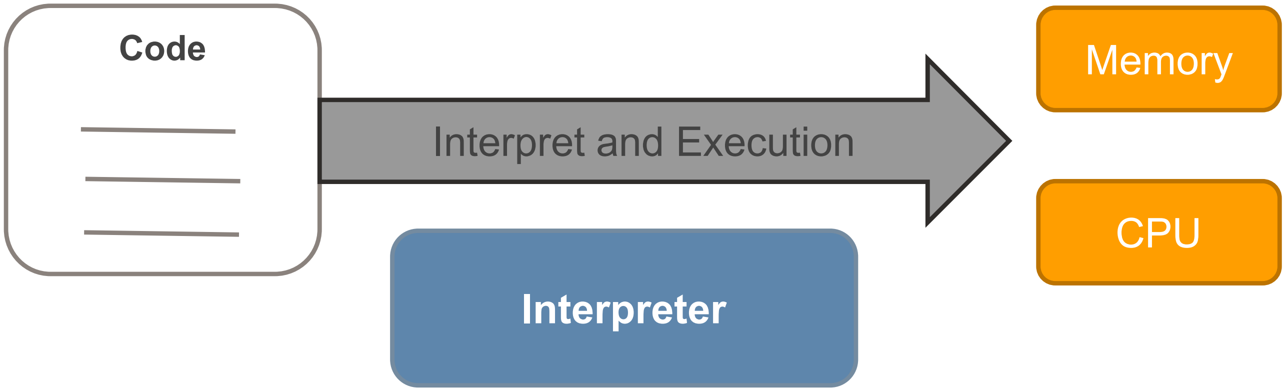
Compiler

- Compiler is a Program that check code syntax and transforms code written in a high-level programming language into the machine code, all at once, before program runs.
- Ex: language use compiler (Compiled) C,C++
- Compiled one time – Run many times



Interpreter

- converts each high-level program statement, one by one, as the flow of the program into the machine code, during program run.
  - Ex: language use Interpreter (Interpreted) JavaScript ,Python
- Every time program needs to run it must be interpreted first

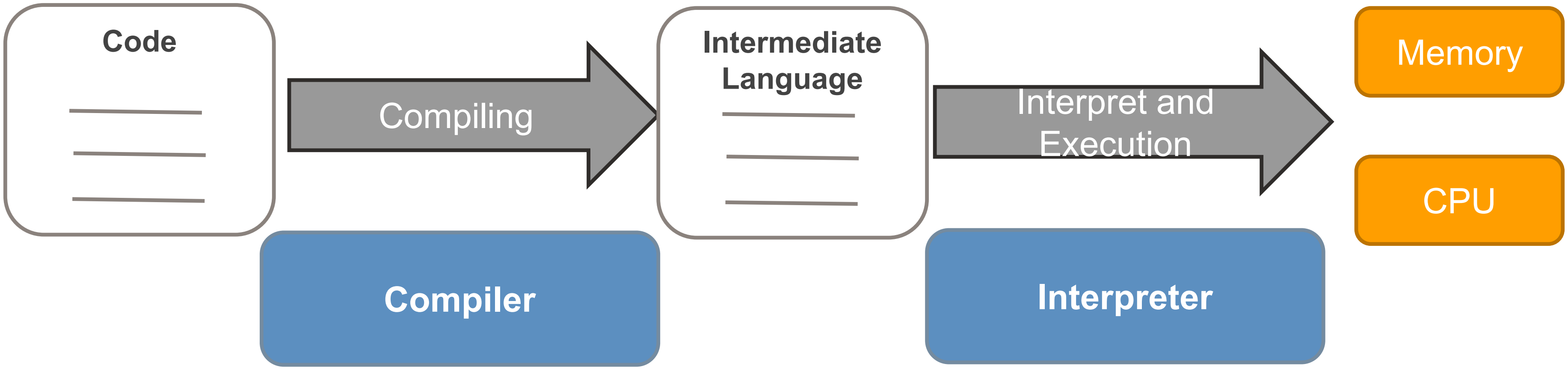


Compiled vs Interpreted

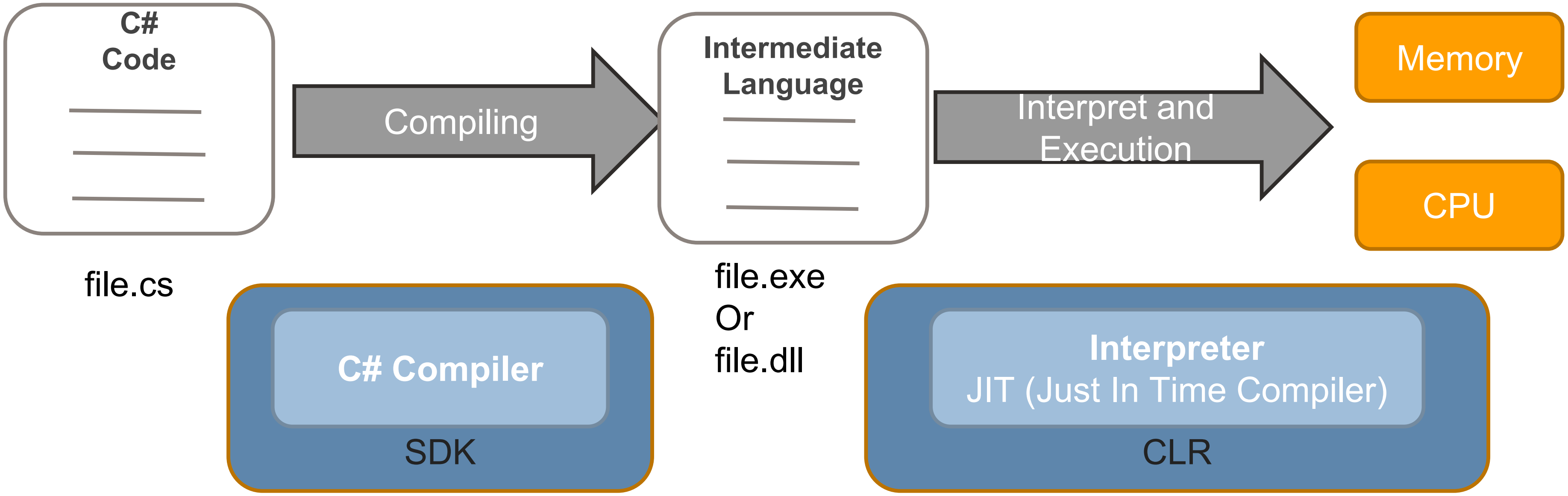
- Since compiler generates machine code the output is related to specific CPU
    - platform dependent
    - Faster in execution
- Since Interpreter execute the code directly it is
    - platform independent
    - Slower in execution

HyperMix

- compiled +Interpreted
  - execution faster than Interpreted
  - Platform Independent
  - Languages uses this concept Java ,C#



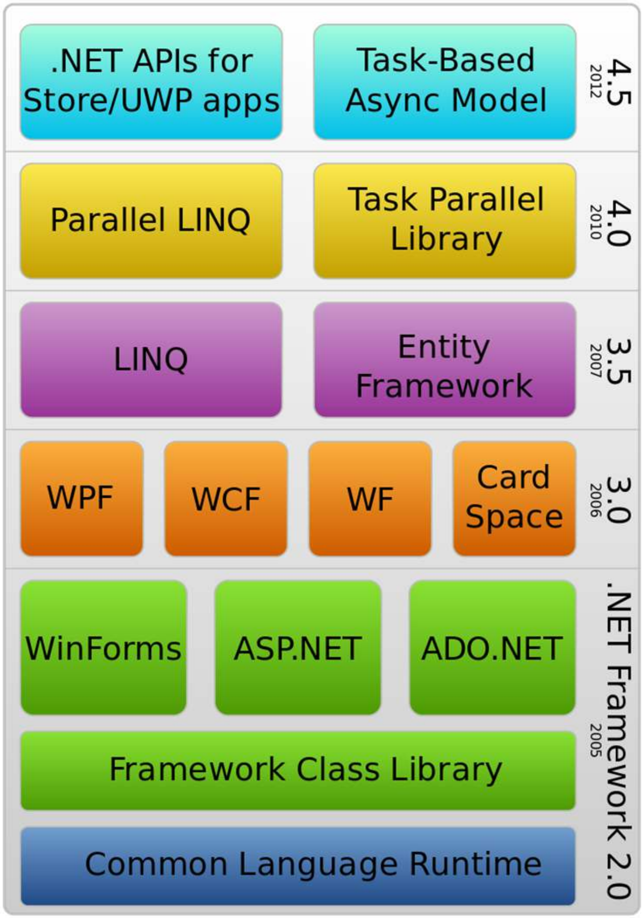
.NET (C#)


C#
Code
file.cs
Compiling
Intermediate
Language
file.exe
Or
file.dll
Interpret and
Execution
Memory
CPU
C# Compiler
SDK
Interpreter
JIT (Just In Time Compiler)
CLR

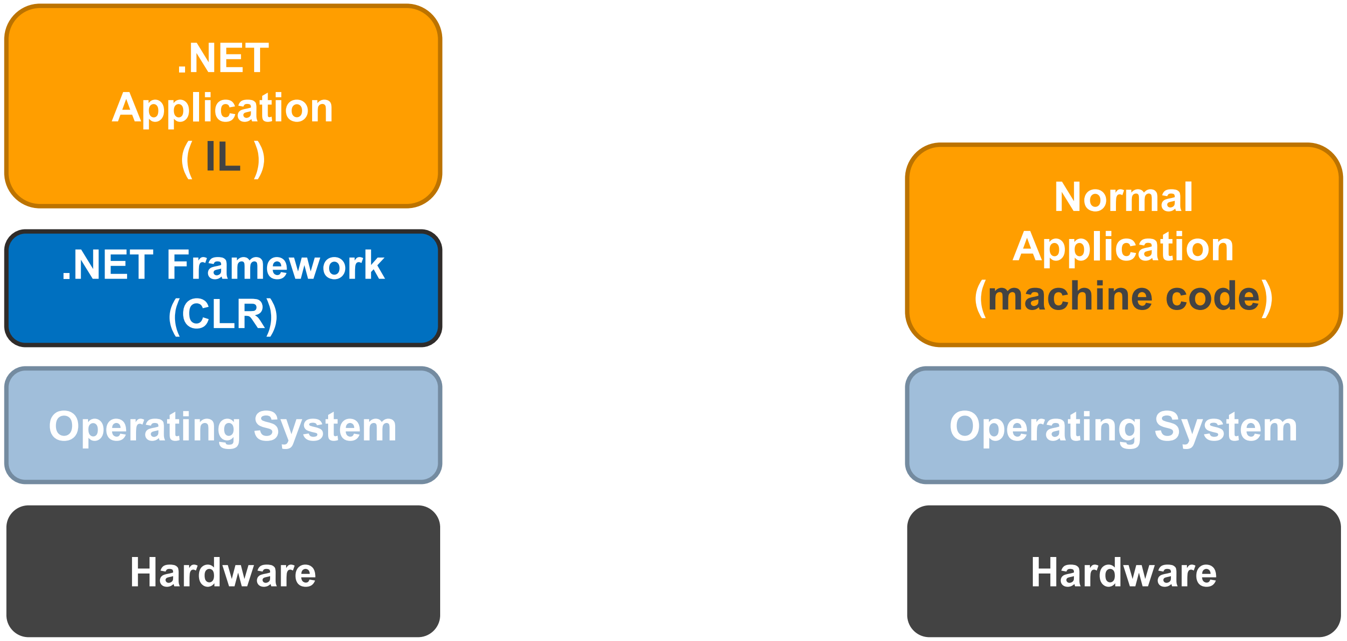
.NET Framework

- .NET framework is software platform developed by Microsoft used to develop various type of applications for *desktop*, *web*, *mobile*, *gaming*
- Mainly consist of two primary components
  - ***CLR*** (common Language Runtime)
    - Runtime Environment
    - Application virtual machine
      - Manage Execution of Code
  - ***Class Library***
    - Set of Libraries
    - Basic class library BCL
      - Contains basic data type
    - Technology class library
      - Contains data types used by specific technology (build specific application)

.NET Framework



Where .NET framework fits



.NET Application (IL)

.NET Framework (CLR)

Operating System

Hardware

Normal Application (machine code)

Operating System

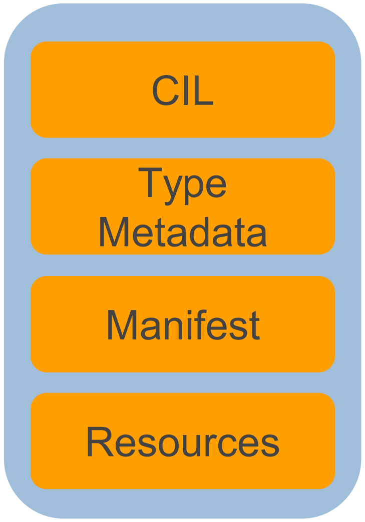
Hardware

Common Language Runtime CLR

- Responsible
  - Executing application
  - Memory Management
  - Security enforcement
  - Language Integration
  - Thread Execution
- Work as ***Operating System*** for .NET application

Components of Assembly

- Generated from the first compiling phase which contain
  - ***CIL*** (Common Intermediate Language)
    - Instruction not specific to certain processor
  - ***Type Metadata***
    - Data about the datatypes within the assembly (name , access levels,….)
  - ***Manifest*** (Assembly Metadata)
    - Which contain the metadata describes
      - Version of the assembly
      - Security Information
      - External assemblies references
      - Exported types
  - ***Resources***
    - Row Data (0’s and 1’s) like image, music,…etc.

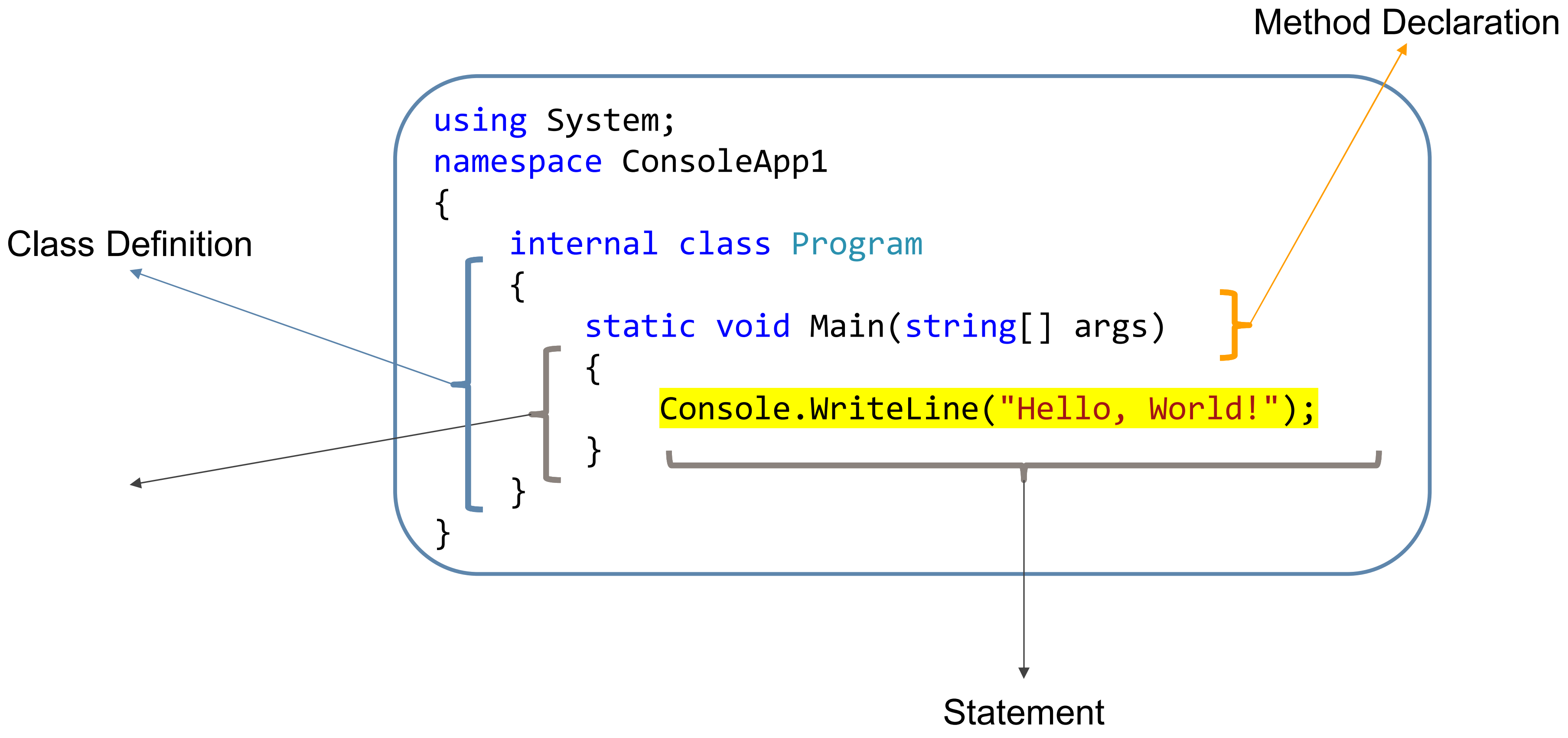


Assembly
(EXE or DLL)

.NET Core (.NET)

- Rewriting .NET framework To make it Platform Independent and open source produce **.NET** (previously named .NET Core)
  - CLR ➔CoreCLR
  - BCL ➔CoreFX
- .NET Core 1 (2016) , 2(2017),3(2019)
- .NET Core 4 skipped (confusing with .NET Framework 4.x)
- .NET 5 (2020)
- .NET 6 (2021) **(LTS)**
- .NET 7.0 (2022)
- Latest Version **.NET 8.0 (2024)** **(LTS-Current)**

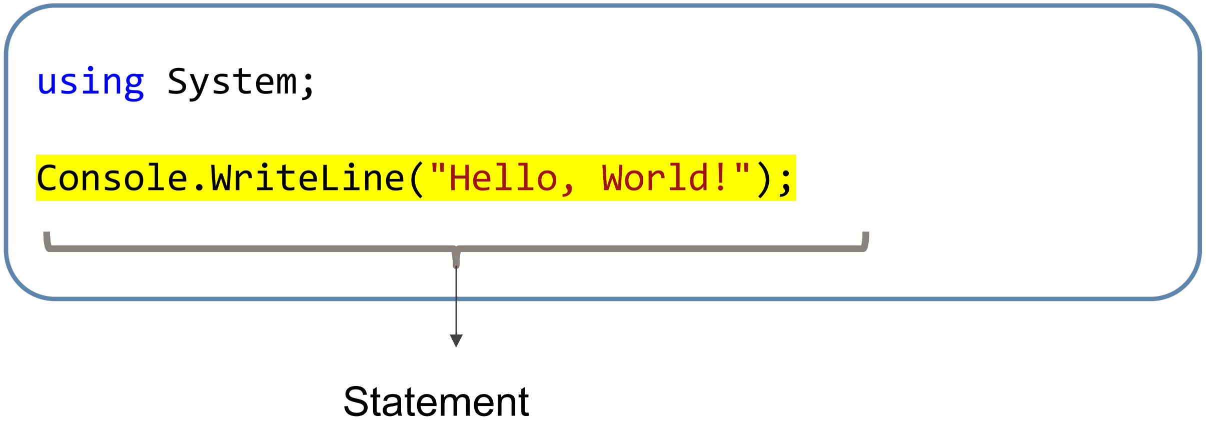
First Program



```
using System;
namespace ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

First Program (top level statement)

- Namespace , **Program** class, **Main** method
  - Generated during compilation
  - Using statements in the beginning of the file
  - Only one file like this



```
using System;

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

Statement

Global using (.NET 6)

- ***global using*** statement
- Show all files➔obj➔net6.0 (or 8.0) ➔ ConsoleApp1.GlobalUsings.g.cs

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Variables and Data Types

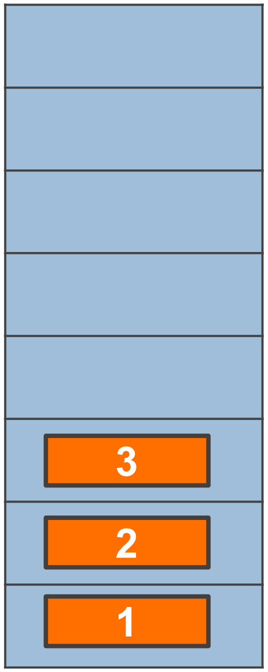
Variables and Data Types

Memory Management

- *CLR* is responsible for memory management it divides memory into two regions (division based on how to treat Memory both are RAM)
  - Stack Memory
  - Heap Memory
- By dividing memory into these two regions, the .NET Framework is able to efficiently manage memory usage and avoid common memory-related issues like stack overflows and heap fragmentation

Stack Memory

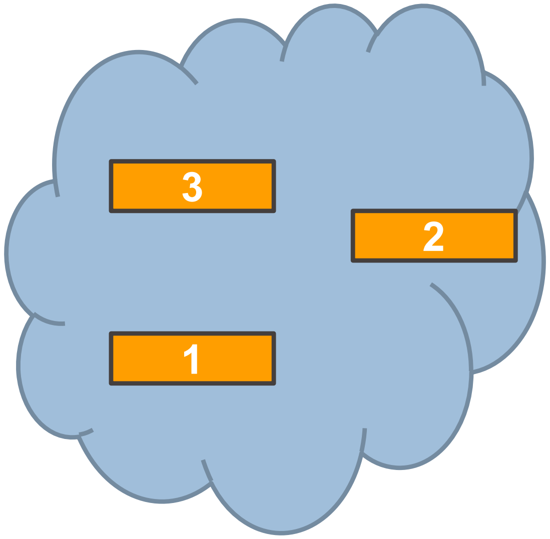
- Stack memory is a special region of memory Used to store ***Small*** variables and ***temporary*** variables created by Methods (*local variables*)
- Data Stored ***Sequential*** (On top of each other)
- ***Limited*** and predetermined at compile-time in size
- Fast Access
- Variables in It can’t be resized



Stack

Heap Memory

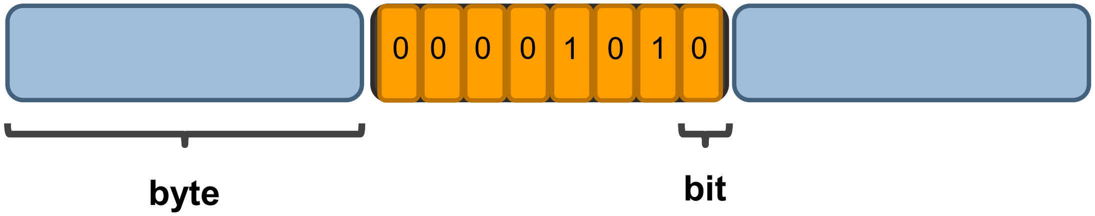
- Heap memory is a region of memory used for ***dynamic*** memory allocation and ***big*** variables and ***global*** variables and variables with long life
- Data Stored ***Scattered*** (collection of memory blocks)
- ***No Size Limits***
- Not Fast access like Stack
- Variables can be resized
- memory is allocated during the execution of instructions written by programmers (*runtime*)
- Managed By ***Garbage collector***



Heap

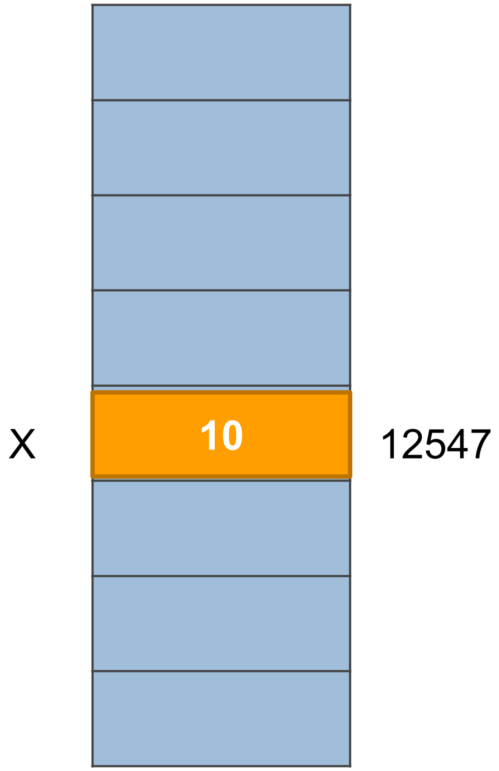
Value Representation in Memory

- every byte in memory has an address
- values represented in memory in binary



What is variable ?

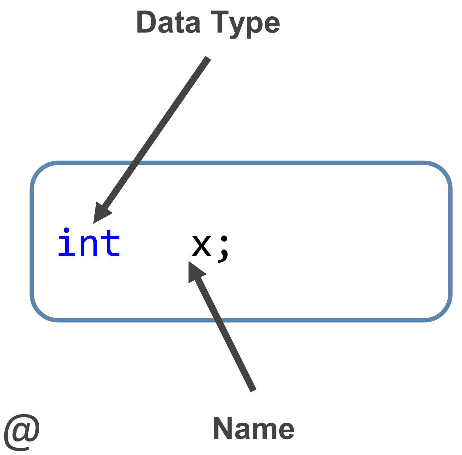
- A memory Location that has
  - Name
  - Size (number of bytes)
  - Address
  - Ex: X =10



Memory

Variable

- Declare variable
  - **Data type**
    - Size of memory location
  - **Name**
    - Must start with a letter
    - Can start or contain _
    - Can't be a digit
    - Can't contain space or symbol like ? , / , - ,*, @

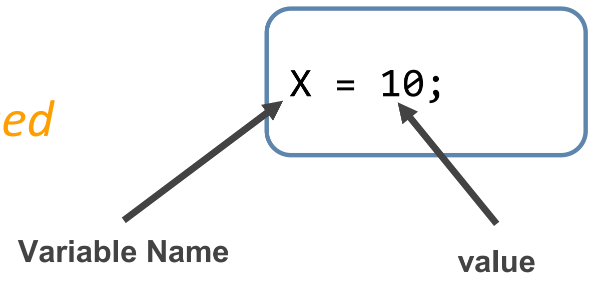


Data Type

```
int   x;
```

Name

- Initialization of a variable
  - Set initial value for the variable
  - Only one time
  - *Variables must be initialized before used*



```
X = 10;
```

Variable Name

value

Naming variables

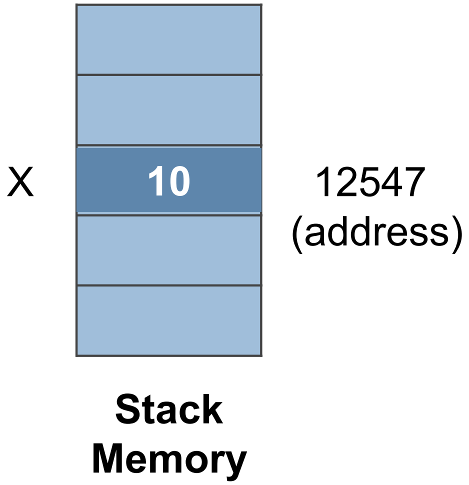
- **Hungarian Notation**
  - In this naming convention the variable name prefix (starts) with group of small letters which indicate data type
  - Ex: **iValue** , **nValue strFirstName** , **txtboxFirstName**
  - Was used by Microsoft in early days of windows programming (Windows API)
- **Camel Notation** or Camel Case
  - In this naming convention first character of all words, except the first word are Upper Case and other characters are lower case.
  - Ex: **firstName, numOfStudents**
  - Used by java , javaScript

Naming variables

- **Pascal Notation** or Pascal Case
  - Similar to Camel but first word start also with upper case Letter
  - Ex: **FirstName** , **NumOfStudents**
  - Used by C# , .NET (in classes and methods)

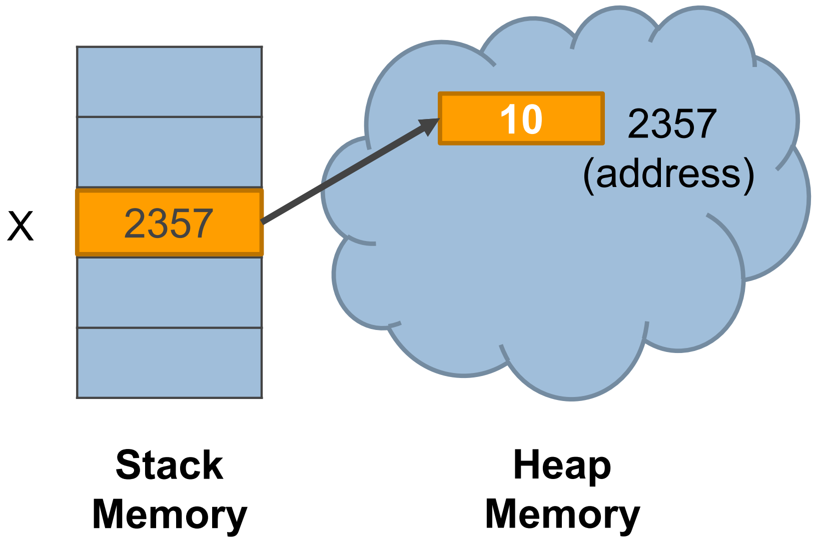
Value Type

- Value type Data Type
  - Stored in the stack memory
  - Variable contains the value itself



Reference Type

- Reference type Data Type
  - Variable refer to value which is in Heap memory
  - Refence variable may stored in Stack or Heap depends on the case (commonly in the Stack memory)



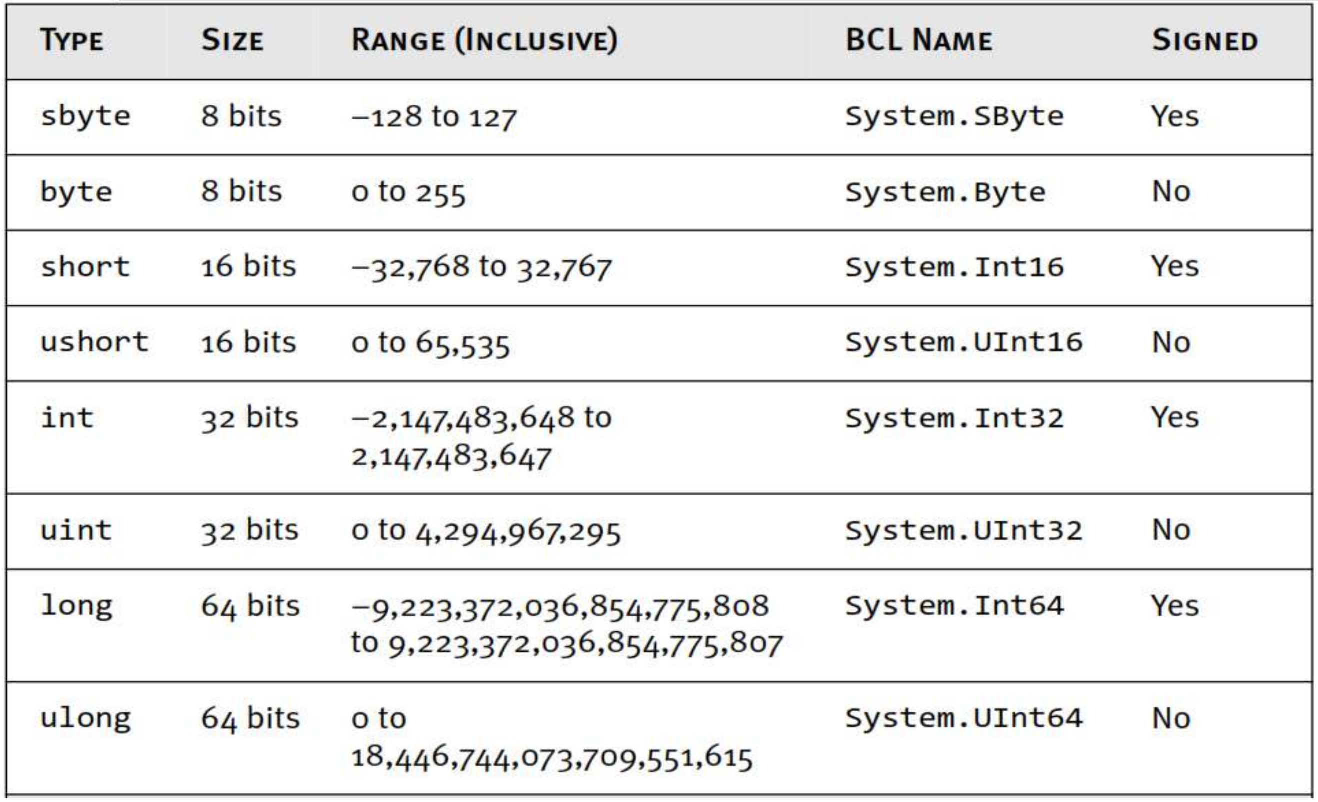
Integer Data Types (value type)

- Special notation
  - Binary notation

```
int x = 0b100;
```

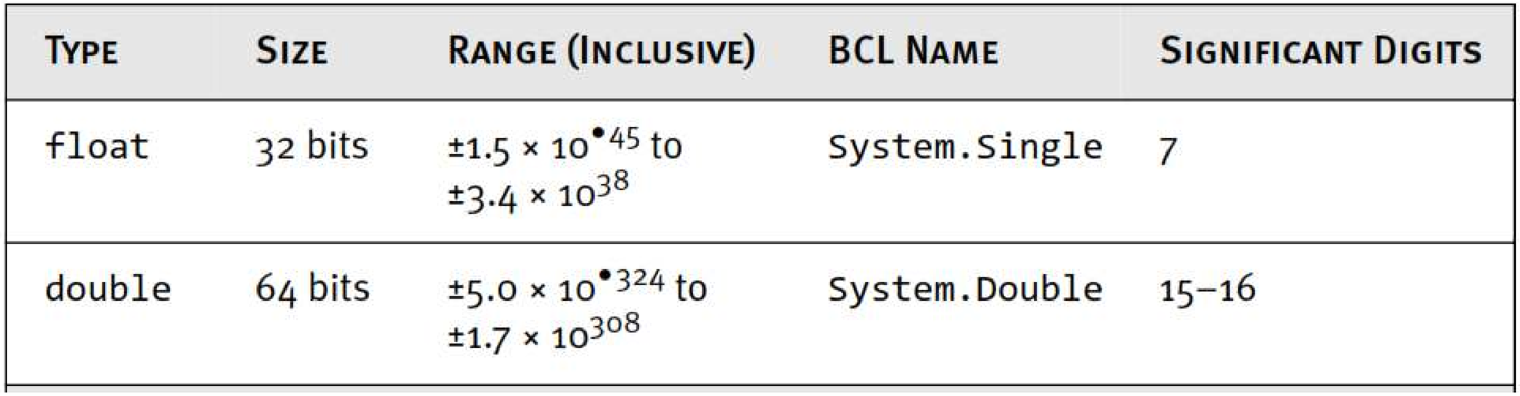
  - Hexadecimal notation

```
int x = 0x100;
```



TYPE	SIZE	RANGE (INCLUSIVE)	BCL NAME	SIGNED
sbyte	8 bits	−128 to 127	System.SByte	Yes
byte	8 bits	0 to 255	System.Byte	No
short	16 bits	−32,768 to 32,767	System.Int16	Yes
ushort	16 bits	0 to 65,535	System.UInt16	No
int	32 bits	−2,147,483,648 to 2,147,483,647	System.Int32	Yes
uint	32 bits	0 to 4,294,967,295	System.UInt32	No
long	64 bits	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	System.Int64	Yes
ulong	64 bits	0 to 18,446,744,073,709,551,615	System.UInt64	No

Floating Point Data Types (value type)

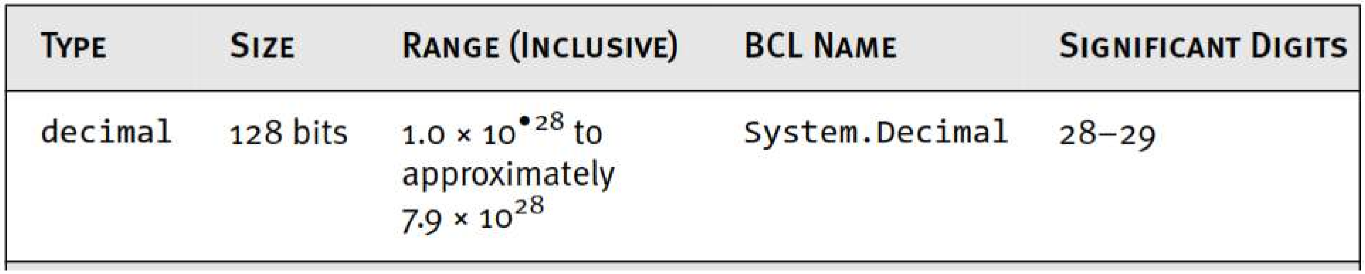


Type	Size	Range (Inclusive)	BCL Name	Significant Digits
float	32 bits	$\pm 1.5 \times 10^{\bullet 45}$ to $\pm 3.4 \times 10^{38}$	System.Single	7
double	64 bits	$\pm 5.0 \times 10^{\bullet 324}$ to $\pm 1.7 \times 10^{308}$	System.Double	15–16

Literal Error

```
float f = 10.5;  // error  to correct it   float f=10.5f;
```

Floating Point Data Types (value type)



TYPE	SIZE	RANGE (INCLUSIVE)	BCL NAME	SIGNIFICANT DIGITS
decimal	128 bits	1.0 × $10^{\bullet 28}$ to approximately 7.9 × 10^{28}	System.Decimal	28–29

Literal Error

```
decimal dm = 10.0;  // error  to correct it   decimal dm=10.0m;
```

value types

- Boolean
  - true , false
  - Ex:

```
bool b = true;
```

- Character
  - Contain one Character
  - Its size = 2 bytes
  - Ex:

```
char ch;
ch = 'A';
```

Reference Types

- String
  - Represent text(multiple characters)

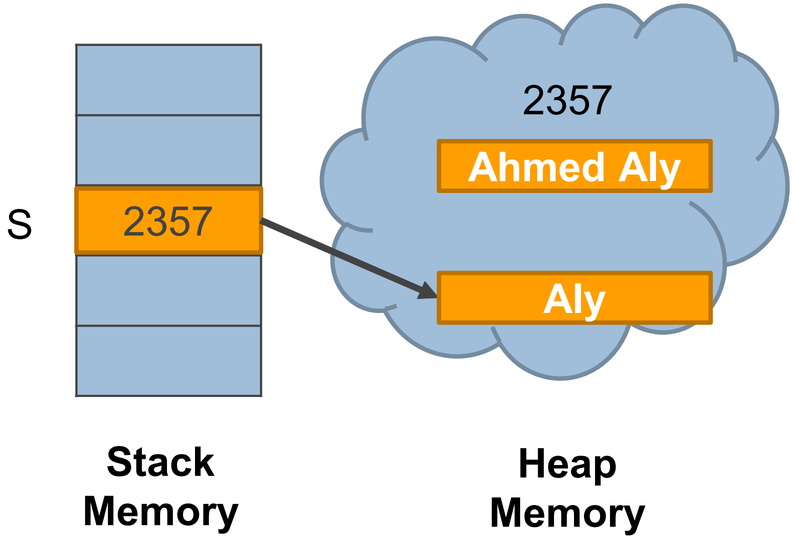
```
string s = "Ahmed";
```

- Array
- Class
- Reference type variables could be Initialized with ***null***

String

- String is a reference type

```
string S = "Ahmed Aly";
Console.WriteLine(s);
```



String

- Declaring and initialization of string

```
// Declare without initializing.
string message1;

// Declare and Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Initialize with a regular string literal.
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";
```

- String is immutable
  - Manipulating methods actually returns new string (does not modified the original string)

String

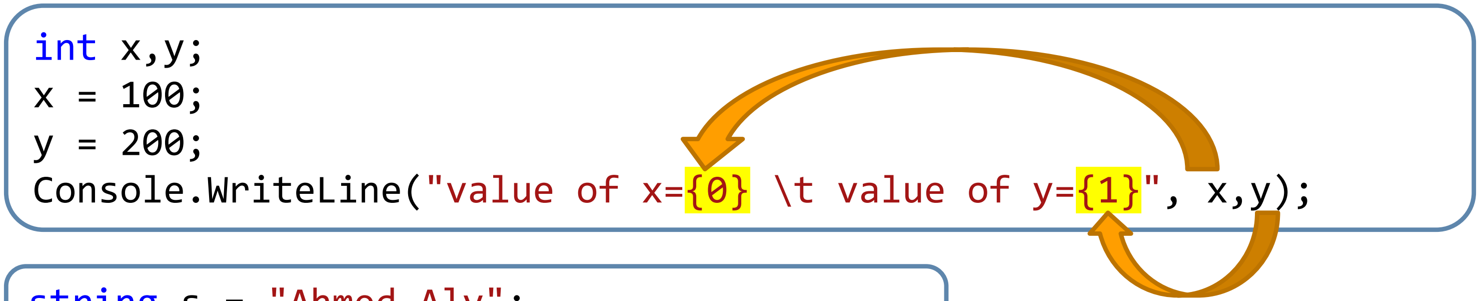
- Methods
  - Static (called through string keyword)
    - Format
    - Concat (Full Name Example)
    - Compare two versions
  - Instance method (called through variable name)
    - StartWith
    - EndWith
    - ToLower
    - ToUpper
    - Trim
    - Replace
    - TocharArray()
    - PadLeft() PadRight()

Input and Output Methods

- Output Methods
  - WriteLine() , Write()
    - Printing Literal string

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

    - Printing value of variable in Literal string



```
int x,y;
x = 100;
y = 200;
Console.WriteLine("value of x={0} \t value of y={1}", x,y);
```

```
string s = "Ahmed Aly";
Console.WriteLine(s);
```

Input and Output Methods

- Output Methods
  - Special Characters

Symbol	Meaning (prints)
\t	Tab spacing
\n	New line
\\	backslash
\’	Single quotes
\”	Double quotation
\r	Carriage return from beginning of the line

Input and Output Methods

- Input Methods
  - **ReadLine**()
    - Reads string from user input
    - Needs user to press Enter Button to finish the process

```
Console.WriteLine("Enter Your Name");
string name=Console.ReadLine();
```

  - **Read**()
    - Reads one character from user Input and return its Unicode number
    - If multiple character reads the first one

```
Console.WriteLine("Enter character");
int code=Console.Read();
```

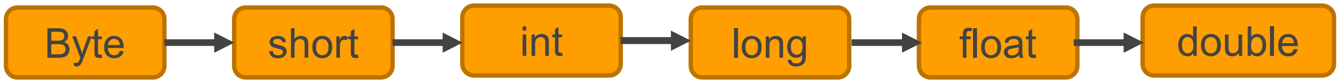
  - **ReadKey**()
    - Reads the keyboard button pressed by the user

Console Helper Methods and Properties

- Methods
  - Console.ResetColor
  - Console.Clear()
  - SetCursorPosition
- Properties
  - Console.BackgroundColor
  - Console.ForegroundColor
  - Console.WindowHeight
  - Console.WindowWidth

Conversion between data types

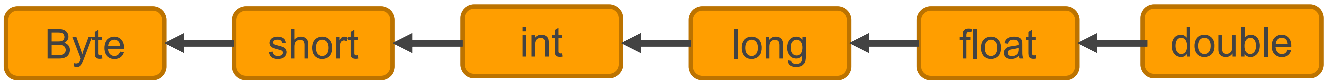
- Implicit Casting
  - For compatible data types (numeric datatypes)
  - Automatic conversion from smaller to bigger



```
int x = 100;
float f;
f = x;
```

Conversion between data types

- Explicit casting
  - For compatible data types (numeric datatypes)
  - It may cause data lost
  - From bigger to smaller



```
float f=3.15f;
int x ;
x = (int) f;
```

Convert to type

Conversion between data types

- Conversion without casting
  - using methods (from string to numbers)
    - Parse ( )

```
Console.WriteLine("Enter Number");
string s = Console.ReadLine();
int x;
x=int.Parse(s);
```

- TryParse ()

```
Console.WriteLine("Enter Number");
string s = Console.ReadLine();
int x;
int.TryParse(s, out x);
```

Conversion between data types

- Conversion without casting
  - using methods
    - ToString ( )

```
int x = 10;
string s = x.ToString();
```

  - ***Convert*** class Methods

```
int x = 10;
string s = Convert.ToString(x);
x= Convert.ToInt32(s);
```

Assignment

- Install Visual Studio (Community Edition)
  - https://learn.microsoft.com/en-us/visualstudio/install/create-an-offline-installation-of-visual-studio?view=vs-2022
    - Download vs bootstrapper (vs_community.exe)
    - For .NET web and .NET desktop development for only one language

```
vs_community.exe  --layout c:\localVSlayout –add Microsoft.VisualStudio.Workload.ManagedDesktop --lang en-US
```

- First program
- Get sum , average for 2 numbers
  - Watch ,breakpoint debug

- .NET SDK (https://dotnet.microsoft.com/en-us/download)
- Install VSCode
  - extension
    - C# Dev Kit extension
    - vscode-solution-explorer
    - Code Runner
      - Settings

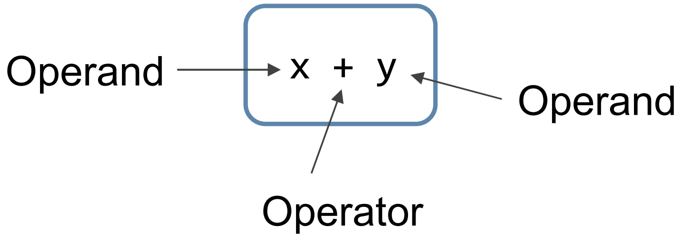
```
□çșḩắsř□□□□çđ□□đîs□□□□độʧŋêʧ□sụŋ□
```

  - Ctrl+Shift+p → .NET Generate Assets for build and debug
  - “launch.json”→ line 17 → `"console": "externalTerminal"`

Operators

Operator

- Operator
  - Some special symbol that tells compiler to perform some action on operands



- C# supports:
  - Unary operators:
    - Requires *one* operand such as x++
  - Binary operators:
    - Requires *two* operands in the expression such as x + 2
  - Ternary operators:
    - Requires *three* operands such as Conditional (? :) operator.

Arithmetic Operators

Operator	Description	Example (y=5)	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment (postfix, prefix)	x=++y	x=6
--	Decrement (postfix, prefix)	x= - -y	x=4

```
int  x,y;
x=y=10;
Console.WriteLine(++x); // print 11
Console.WriteLine(y++); // print 10
```

Assignment Operators

Operator	Example	Same As	Result (y=10)
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise Left Shift
>>	Bitwise Right Shift

```
int a = 60;/* 60 = 0011 1100 */
int b = 13;/* 13 = 0000 1101 */
int c = 0;

c = a & b; /* 12 = 0000 1100 */
Console.WriteLine("Line 1 - Value of c is {0}", c);

c = a | b; /* 61 = 0011 1101 */
Console.WriteLine("Line 2 - Value of c is {0}", c);

c = a ^ b; /* 49 = 0011 0001 */
Console.WriteLine("Line 3 - Value of c is {0}", c);

c = ~a;   /*-61 = 1100 0011 */
Console.WriteLine("Line 4 - Value of c is {0}", c);

c = a << 2;/* 240 = 1111 0000 */
Console.WriteLine("Line 5 - Value of c is {0}", c);

c = a >> 2;/* 15 = 0000 1111 */
Console.WriteLine("Line 6 - Value of c is {0}", c);
```

Comparison Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equality
!=	Inequality

Logical Operators

Operator	Description
&&	Logical "AND" – returns true when both operands are true; otherwise it returns false
||	Logical "OR" – returns true if either operand is true. It only returns false when both operands are false
!	Logical "NOT"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand

Precedence and Associativity

- ***Operator Precedence***: Determines the order in which operators are evaluated. Operators with higher precedence are evaluated first.
- ***Operator Associativity***: Determines the order in which operators of the same precedence are processed.

  - In an expression with multiple operators, the operators with higher precedence are evaluated before the operators with lower precedence
  - Ex:

```
int a = 2 + 2 * 2;
Console.WriteLine(a); // 6
```

```
int a = (2 + 2) * 2;
Console.WriteLine(a); // 8
```

Precedence and Associativity

- Precedence from high to low

Category	Operator	Associativity
Postfix	( ),[ ] ,++,--	Left To Right
unary	--,++,!	Right to Left
mutilation	* , / , %	Left to Right
Addition	+ , -	Left to Right
Shift	>>, <<	Left to Right
Relational	<,<=,>,>=	Left to Right
Equality	==,!=	Left to Right
Logical and	&&	Left to Right
Logical Or	||	Left to Right
Conditional	?:	Right to Left
Assignment	=, +=, -=, *=, /=, %=	Right to Left

Assignment

- Get sum , average for 2 numbers entered by the user