

# C#

## (Language Fundamentals)

# A simple C# program

```
using System;

class HelloClass{

    public static int Main() {

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

        return 0; }}


```

Variations on Main() method→

```
public static void Main(string[] args)
```

```
public static void Main()
```

```
public static int Main()
```

The HCL logo is displayed in a bold, blue, sans-serif font.

# Saving, Compiling and Executing the Program

- Save the file with extension “.cs” .(Test.cs)
- Goto command window and goto the directory where the compiler is(csc-c sharp compiler)
- Compile the file :
  - csc Test.cs
- After successful compilation .EXE file is created(Test.EXE)
- Execute the EXE file

# First steps

```
using System;  
  
class HelloWorldClass{  
  
    public static int Main()  
    {  
        Console.WriteLine("Hello World!!!");  
        return 0;  
    }  
}
```

Explained at the end of this session

Execution starts from here

Writes Hello World!!! on the console

C# requires you put all the code inside a construct called *class*.

# Formatting console output

```
using System;
```

```
class HelloClass{
```

```
    public static void Main(string[] args){
```

```
        int intval=90;
```

```
        double doubleval=9.99;
```

```
        bool boolval=true;
```

```
        Console.WriteLine("Int value= {0} \n Double value= {1}
```

```
        \n Boolean value= {2} ", intval, doubleval,boolval);
```

```
    }
```

```
}
```

```
Console.WriteLine("Int value="+intval +" \n
```

```
Double value= "+ doubleval +" \n Boolean
```

```
value= +boolval);
```

## string formatting flags

String format character	description
C or c	Format currency
D or d	Format decimal numbers
E or e	Exponential notation
F or f	Fixed-point formatting
G or g	Formats a number to fixed or exponential format
N or n	Used for basic numerical formatting (with commas)
X or x	Used for hexadecimal formatting

- `Console.WriteLine("C format:{0:C} ", 999.982);`
- `Console.WriteLine("D9 format:{0:D3} ", 35);`
- `Console.WriteLine("E format:{0:E} ", 999.982);`
- `Console.WriteLine("F3 format:{0:F3} ", 999.982);`
- `Console.WriteLine("N format:{0:N} ", 999.982);`
- `Console.WriteLine("X format:{0:X} ", 10);`
- `Console.WriteLine("G format:{0:G} ", 999.982);`

```
C format: Rs. 999.98
D9 format: 035
E format: 9.999820E+002
F3 format: 999.982
N format: 999.98
X format: A
G format: 999.982
```

# Reading inputs from the console

```
using System;
class SayHello{
public static void Main(){
    string s;
    Console.Write("Please your name");
    s = Console.ReadLine();
    Console.WriteLine("Hello "+s);
}
}
```



# The System Data Types

C# type	CLS compliant	System Type	Range
sbyte	No	<code>System.Sbyte</code>	-128 to 127 (signed 8-bit)
byte	Yes	<code>System.Byte</code>	0 to 255 ( unsigned 8 bit)
short	Yes	<code>System.Int16</code>	-32768 to 32767 (signed 16 bit )
ushort	No	<code>System.UInt16</code>	0 to 65535 (unsigned 16 bit)
int	Yes	<code>System.Int32</code>	-2147783648( $-2^{31}$ ) to 2147483647( $2^{31}-1$ ) (signed 32 bit)
uint	No	<code>System.UInt32</code>	0 to $2^{32}-1$ (unsigned 32 bits)

C# type	CLS compliant	System Type	Range
long	Yes	<b>System.Int64</b>	$-2^{63}$ to $2^{63}-1$ (signed 64 bit number)
ulong	No	<b>System.UInt64</b>	0 to $2^{64}-1$ (unsigned 64 bit number)
char	Yes	<b>System.Char</b>	U0000 to Uffff (16 bit unicode character)
float	Yes	<b>System.Single</b>	$1.5 \times 10^{-45}$ to $3.4 \times 10^{38}$ (32 bit floating point number)
double	Yes	<b>System.Double</b>	$5.0 \times 10^{-324}$ to $1.7 \times 10^{308}$ (64 bit floating point number)
bool	Yes	<b>System.Boolean</b>	true or false
decimal	Yes	<b>System.Decimal</b>	$10^0$ to $10^{28}$ (96-bit signed number)
string	Yes	<b>System.String</b>	Limited by system memory (Represents a set of unicode characters)
object	Yes	<b>System.Object</b>	Any type can be stored in an object variable (Base class of all types in .NET)

# Introducing Array Types

- Array declaration and initialization
  - `int[] n1={20,10,5,13};`
- Accessing array elements
  - `n1[0]`
- Length of an array
  - `n1.Length`

## NOTE:

1. `int n1[]={20,10,5,13};` → error

2. `int[5] n1;` → error

3. `int[] n1 ; n1[0]=4;` → error

4. *More on arrays in the next topic*

# Literals

- Literal is a value that can be assigned to a variable.
- Types
  - Boolean literals
  - Integer literals
  - Floating-point literals
  - Character literals
  - String literals

# Boolean literals

- Type: `bool`

- Values

- `true`

- `false`

- Example

```
bool correct = true;
```

```
bool wrong = false;
```

# Integer literals

- Integer literals can be written in two ways
  - Decimal: Example: 345, 2345678901
  - Hexadecimal where decimal numbers from 0 to 15 is represented as 0-9A-F: Example 0x11, 0XFF
- Integer literals types
  - Literal with no suffix (example 25) → could be **int**, **uint**, **long**, **ulong**
  - Literal with u or U as suffix (25u, 35U) → could be **uint** or **ulong**

- Literal with l or L as suffix (6544425L, 76l) → could be **long** or **ulong**
- Literal with ul or or uL or Ul or UL as suffix (25ul, 35UL) → is **ulong**
- Any value outside the **ulong** range generates compilation error

# Floating-point literals

- Floating-point literals can be written in two ways
  - Fixed notation: 3.14
  - Scientific notation: 0.314E1, 314e-2
- Floating-point literal types
  - Literal with no suffix → is **double**
  - Literal with F or f as suffix (Example 3.14f, 3.14F) → is **float**
  - Literal with D or d as suffix (Example 3.14D, 3.14D) → is **double**
  - Literal with M or m as suffix (Example 3.14m, 3.14M) → is **decimal**



# Guess why?

- Following code generates error. Why?

```
using System;  
class Fix{  
public static void Main(){  
float f=0.314e1;  
Console.WriteLine(f);  
}  
}
```

# Character literals

- Character literals can be written in the following forms
  - Any character within single quotes : Example '1','a'
  - Unicode escape sequence: Example '\u0041' represents the character 'A' in unicode
  - Hexadecimal escape sequence: Example '\x41' represents the character 'A' in unicode
  - Any escape sequence: Example \n for new-line character

# Escape Characters

- `\'` -inserts a single quote into string literal
- `\"` -inserts a double quote
- `\\` -inserts a backslash into string literal
- `\a` -triggers system alert(beep)
- `\n` -inserts a new line
- `\r` -inserts a carriage return
- `\t` -inserts a horizontal tab into the string literal

# String literals


- String literals can be written in two ways
  - Regular string literal (enclosed within double quotes): Example: `"C#", "C:\\My\\test"`
  - Verbatim string literals (@ preceding regular string literal)
  - @ tells the compiler to interpret the character that appears between the double quote in the exactly the same way as they are written
  - Example: `@"C:\\My\\test"`, `@" This is Madam's Pen"`

# Solve

- How can you correct the program given below?

```
using System;  
class Str{  
public static void Main(){  
string s="Mile to go  
        before i sleep";  
  
Console.WriteLine(s);  
}}
```

New-line  
character inserted  
intentionally in the  
code



# Operators

## Relational Operators

==	!=	<	>	>=	<=
----	----	---	---	----	----

## Logical Operators

&&		!	&		^	~
----	--	---	---	--	---	---

## Arithmetic Operators

+	-	/	*	++	--	%
---	---	---	---	----	----	---

## Shift Operators

>>	<<
----	----

# Assignment Operators

=	>=	<=	*=	/=	&=	=	^=	<<=	>>=	%=
---	----	----	----	----	----	---	----	-----	-----	----

## Other Operators

?:

[]

()

??

::

.

is

as

typeof

new

checked

unchecked

# Checked and unchecked operator

```
byte b =255;  
b++;
```

- Incrementing the value of b causes overflow since byte can hold values in the range 0 to 255 only.
- The value of b is 0 finally.
- To make the CLR **throw an error at runtime** when overflow occurs, the above code can be written inside the checked block.

```
byte b =255;  
checked{ b++; }
```

- Right opposite to checked operator is unchecked operator. Unchecked is the default behaviour.



# Looping Statements

for loop:

```
for (initialization;  
    condition; increment/decrement) {  
    //statements  
}
```

Ex: `for (int i=0; i<10; i++)`

foreach loop:

```
foreach (object var in itemlist) {  
    //statements  
}
```

Array or Collection

Ex: `int[] arr={1,2,3,4,5};  
 foreach (int s in arr)  
 Console.WriteLine(s);`

## While loop

```
while(Condition) {  
    statements  
}
```

Ex: `int i=0;`

```
while(i<10) {  
    Console.WriteLine(i);  
    i++; }  
}
```

Condition must evaluate to bool value



## do-while loop

```
do{  
    statements  
}while(Condition);
```


Ex: `int i=0;`

```
do {  
    Console.WriteLine(i);  
    i++;  
} while(i<10);
```

Condition must evaluate to bool value



# break and continue

- Used with loop statements
  - break is used to break out of the loop. Is used with switch statement also.
  - continue is used to exit out of current iteration and continue with the next iteration.
- 
- ahead

# Decision constructs

## if/else statement

```
if (Condition) {  
    statements;    }  
else {  
    statements;    }
```

Condition must evaluate to bool value

## switch statement

```
switch (var) {  
    case val1: statements;  
                break;  
    case val2: statements;  
                break;  
    .....  
    default: statements;  
                break;    }
```


Any numeric value or string

# Break the switch-case

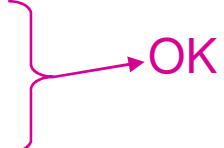
- C# compiler enforces that every case statement must have a break statement. In other words control cannot fall through from one case label to another.

```
switch (color) {  
    case 'r': paintRed();  
                break;  
    case 'g': paintGreen();  
    case 'b': paintBlue();  
                break;  
}
```

Missing  
break  
statement



```
switch (color) {  
    case "red":  
    case "RED":  
        paintRed();  
        break;  
    case "green":  
    case "GREEN":  
        paintGreen();  
        break;  
}
```



*But what if I want fall-through ???*

# Fall through

- One exception when **break** can be omitted in the **case** statement is when there is a **goto** statement.

- `goto case caselabel`  

```
switch(s[0]) {  
    case "Red":  
    case "red": goto case "RED";  
    case "Green": Console.WriteLine("Green");  
                break;  
    case "RED": Console.WriteLine("Red");  
                break;  
    default: Console.WriteLine("Bye");  
            break;  
}
```

```
using System;  
class Switch{
```

```
    public static void Main() {  
        string s;  
        int num;  
        int pow;
```

**start:**

```
        Console.Write("Please enter a number");  
        s = Console.ReadLine();  
        num = Int32.Parse(s);
```

→ Converts string to int.  
More like these we will  
discover as we move on...

```
        Console.Write("Enter the power (1 to 3)");  
        s = Console.ReadLine();  
        pow = Int32.Parse(s);
```

**HCL**

```
switch (pow) {  
    case 1:  
        Console.WriteLine("Power is {0}.", num);  
        break;  
    case 2:  
        Console.WriteLine("Power is {0}.", num *  
num);  
        break;  
    case 3:  
        Console.WriteLine("Power is {0}.",  
num*num*num);  
        break;  
    default:  
        Console.WriteLine("Your number is not  
between 1 and 3.");  
        break;  
}
```



```
decide:
```

```
Console.WriteLine("Type \"cont\" to continue or  
\"stop\" to stop: ");
```

```
    s = Console.ReadLine();
```

```
    switch (s)    {
```

```
        case "cont":
```

```
            goto start;
```

```
        case "stop":
```

```
            Console.WriteLine("Bye.");
```

```
            break;
```

```
        default:
```

```
            Console.WriteLine("Incorrect input.");
```

```
            goto decide;
```

```
    }
```

```
}
```

```
}
```

# Conversions

- Implicit conversion
  - Happen automatically
- Explicit conversion
  - Cast need to be called
- All the cast must be static (known at compile-time).

# Implicit Conversions

- Identity conversions
- Implicit numerical conversions
- Implicit enumeration conversions
- Implicit reference conversions
- Boxing conversions
- Implicit type parameter conversions
- Implicit constant expression conversions
- User-defined conversions

→ enum topic

→ inheritance topic

# Identity conversions

- Involves conversion from one type into the same type.

- Example: `int k;`

`int j=k;` → identity conversion happen automatically

# Implicit numerical conversions

- `sbyte` → `short`, `int`, `long`, `float`, `double`, `decimal`
- `char` → `int`, `long`, `float`, `double`, `decimal`, `ushort`, `uint`, `ulong`
- `byte` → `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `float`, `double`, `decimal`
- `short` → `int`, `long`, `float`, `double`, `decimal`
- `ushort` → `int`, `uint`, `long`, `ulong`, `float`, `double`, `decimal`
- `int` → `long`, `float`, `double`, `decimal`
- `uint` → `long`, `ulong`, `float`, `double`, `decimal`
- `ulong`, `long` → `float`, `double`, `decimal`
- `float` → `double`

Note that some of the above conversion will cause loss of precision but not loss of magnitude.

```
using System;
class Convert{
public static void Main() {
char c='A';
int i=c;
Console.WriteLine(i);

long l=23456789100;
float f=l;
Console.WriteLine(f);

}}
```

Note vice versa is not allowed implicitly

Prints:

65

2.345679E+10

*Loss of precision !*

**HCL**

# Processing Command Line Arguments

```
using System;
```

```
class HelloClass{
```

```
    public static int Main(string[] args) {
```

```
        for (int i=0; i<args.Length; i++)
```

```
            Console.WriteLine("Arg {0}", args[i]);
```

```
        return 0;
```

```
    }
```

Convert the above for-loop to for-each loop.

```
}
```

Length property of System.Array  
checks if there is any item in the  
string array.

# Conversion and Casting

- Implicit conversion

- Automatic

```
int i1 = 5;
```

```
long l1 = i1;
```

- Casting

- Explicitly conversion

```
long l2 = 12500;
```

```
int i2 = (int)l2;
```



# Implicit conversion list

Source Type	Target Type
Byte	short, ushort, int, uint, long, ulong, float, double, or decimal
Sbyte	short, int, long, float, double, or decimal
Int	long, float, double, or decimal
UInt	long, ulong, float, double, or decimal
Short	int, long, float, double, or decimal
Ushort	int, uint, long, ulong, float, double, or decimal
Long	float, double, or decimal
Ulong	float, double, or decimal
Float	double

# Explicit conversion list

Source Type	Target Type
Byte	sbyte or char
Sbyte	byte, ushort, uint, ulong, or char
Int	sbyte, byte, short, ushort, uint, ulong, or char
UInt	sbyte, byte, short, ushort, int, or char
Short	sbyte, byte, ushort, uint, ulong, or char
Ushort	sbyte, byte, short, or char
Long	sbyte, byte, short, ushort, int, uint, ulong, or char
Ulong	sbyte, byte, short, ushort, int, uint, long, or char
Float	sbyte, byte, short, ushort, int, uint, long, ulong, char, or decimal
Double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or decimal
Char	sbyte, byte, or short
Decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or double

# Namespaces

- Namespaces are used to group the related type.

*syntax for creating a namespace;*

```
namespace mynamespace
{
    //class definitions
}
```

```
using System;
using StudentSpace.s1;
class MainClass {
public static void Main()      {
    Student.display();
}
}
namespace StudentSpace.s1 {
    class Student{
        static string[] names
        ={"Vani", "Vinod", "Vibhav",
        "Varun", "Vishal", "Vidya"};
        public static void display() {
            foreach(string s in names)
                Console.WriteLine(s);
        }
    }
}
```

# Using namespace alias

- The display function can be called in another way using namespace alias .

```
using System;  
using ss = StudentSpace.s1.Student;  
class MainClass {  
  
    public static void Main()    {  
        ss.display();  
    }  
  
}
```

# Nesting namespace

- In C#, namespaces can be nested with each other as showing below.

- ```
using System;
namespace Outer{
namespace Inner{
class MyClass{
public MyClass(){
{Console.WriteLine("My Class");}
}}
}
class MainClass{
public static void Main(){
Outer.Inner.MyClass mc = new
Outer.Inner.MyClass();
}
}
```