Chapter-3 Elements of .Net Languages

What are Data Types in C#?

- The data type tells the C# compiler what kind of value a variable can hold. C# includes many in-built data types for different kinds of data
- E.g. String, integer, float, decimal, etc.
- A variable must be declared with the data type because C# is a strongly-typed language. For example

  String msz = "HELLO WORLD";

Here, ***String*** is a data type, ***msz*** is a variable, and *"HELLO WORLD"* is a string value assigned to a variable ***msz***.

- The C# language comes with a set of Basic data types and these data types are used to build values which are used within an application.

Example:

```
static void Main(string[] args)
    {
        string stringVar = "Hello World!!";
        int intVar = 100;
        float floatVar = 10.2;
        char charVar = 'A';
        bool boolVar = true;
    }
```

- Each data types includes specific range of values

- ***Data types in C# is mainly divided into three categories***

  1. Value Data Types
  2. Reference Data Types
  3. Pointer Data Type

1. Value Data Type:

- Value Data Types, directly stores the variable value in memory and it also accept both signed and unsigned literals.
- The derived class for these data types are *System.ValueType*.
- Following are different Value Data Types

I) Signed & Unsigned Integral Types:

- There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.

Signed Integral

- ✓ These data types hold integer values (both positive and negative).
- ✓ Out of the total available bits, one bit is used for sign.
- ✓ Sbyte, Short, int, long.

Unsigned Integral

- ✓ These data types only hold values equal to or greater than 0.
- ✓ We generally use these data types to store values when we are sure, we won't have negative values.
- ✓ Byte, ushort, uint,ulong

II) Floating Point Types :

- These data types hold floating point values i.e. numbers containing decimal values. For example, 12.36, -92.17, etc.
- There are 2 floating point data types

a) Float:

- It is **32-bit single-precision** floating point type.
- It has 7 digit Precision. And 32 bits size.
- To initialize a float variable, use the suffix f or F.
- Example float x = 2.5F;.
- Default value: 0.0F
- F at the end represent the value of float type.

- If the suffix F or f will not use then it is treated as double.

```
class FloatExample
    {
        public static void Main(string[] args)
        {
            float number = 143.27F;
            Console.WriteLine(number);
        }
    }
```

O/P

143.27

b) Double:

- It is **64-bit double-precision** floating point type.
- It has 14 – 15 digit Precision. And 64 bits size.
- To initialize a double variable, use the suffix d or D.
- But it is not mandatory to use suffix because by default floating data types are the double type

```
class DoubleExample
    {
        public static void Main(string[] args)
        {
            double value = -11092.53D;
            Console.WriteLine(value);
        }

    }
```

O/P

-11092.53

III) Character Types :

- It represents the 16-bit Unicode character.
- Name - char
- Size – 16 bits
- Range- U+0000 to U+ffff
- Default value : '\0'

Example:

```
class CharExample
        {
            public static void Main(string[] args)
            {
                char ch1 = '\u0042';
                char ch2 = 'x';
                Console.WriteLine(ch1);
                Console.WriteLine(ch2);
            }
        }
```

O/P

B

x

How? → The Unicode value of 'B' is '\u0042', hence printing ch1 will print 'B'

IV) Decimal Types :

- Decimal type has more precision and a smaller range as compared to floating point types (double and float). So it is appropriate for monetary calculations.
- Name: Decimal
- **Size**: 128 bits
- **Default value**: 0.0M [M at the end represent the value is of decimal type]
- It has 28-29 digit Precision.
- To initialize a decimal variable, use the suffix m or M.
- Example, decimal x = 2500.25m;.
- The suffix M or m must be added at the end otherwise the value will be treated as a double and an error will be generated.

```
class DecimalExample
        {
            public static void Main(string[] args)
            {
                decimal bankBalance = 53005.25M;
                Console.WriteLine(bankBalance);
            }
        }
```

O/P

53005.25

V) Boolean Types :

- It has to be assigned either true or false value.

- ❖ Values of type bool are not converted implicitly or explicitly (with casts) to any other type.
- ❖ But the programmer can easily write conversion code
- ❖ Name – bool
- ❖ Values- T or F
- ❖ Default value-F

- ❖ Boolean are generally used to check conditions such as in *if statements*, *loops*, etc.

Exampel:

```
namespace DataType
{
    class BooleanExample
    {
        public static void Main(string[] args)
        {
            bool isValid = true;
            Console.WriteLine(isValid);
        }
    }
}
```

O/P

True

2. Reference Data Types :

- ❖ It contains a memory address of variable value because the reference types doesn't store the variable value directly in memory.
- ❖ The built-in reference types are ***string, object.***

String

- ✓ Represents a sequence of Unicode characters
- ✓ Its type name is **System.String**.
- ✓ string and String are equivalent.
- ✓

```
static void Main(string[] args)
        {
            String message = "Hello";
            Console.Write(message);

            Console.ReadKey();
        }
```

string s1 = "WELCOME TO"; // created through string keyword

String s2 = "EEC COLLEGE"; // created through String class

Object:

- ✓ In C#, all types (predefined and user-defined, reference types and value types) inherit directly or indirectly from Object.
- ✓ So basically it is the base class for all the data types in C#.
- ✓ Before assigning values, it needs type conversion.
- ✓ When a variable of a value type is converted to object, it's called **boxing**.
- ✓ When a variable of type object is converted to a value type, it's called **unboxing**.
- ✓ Its type name is **System.Object**.

3. Pointer Data Type

- ❖ It contains a memory address of the variable value.
- ❖ To get the pointer details we use two symbols ***ampersand (&) and asterisk(*)***
- ❖ **ampersand (&)** →known as Address Operator and used to determine the address of a variable
- ❖ ***asterisk (*)*** → also known as Indirection Operator and used to access the value of an address.

Example: int* p1, p; // Valid syntax

int *p1, *p; // Invalid

SUMMARY:

Reserved Word	.NET Type	Type	Size (bits)	Range (values)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65,535
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
long	Int64	Signed integer	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	UInt64	Unsigned integer	64	0 to 18,446,744,073,709,551,615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	(+ or -)1.0 x 10e-28 to 7.9 x 10e28
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or False
object	Object	Base type of all other types		
string	String	A sequence of characters		
DateTime	DateTime	Represents date and time		0:00:00am 1/1/01 to 11:59:59pm 12/31/9999

Conversion / Type Conversion

- ❖ The values of certain data types are automatically converted to the different data types in C#. This is called implicit conversion.
- ❖ The following example demonstrates an implicit conversion.

Example: Implicit Conversion

```
static void Main(string[] args)
        {
            int i = 345;
            float f = i;

            Console.WriteLine(f);

        }
```

- ❖ In the above example, value of an integer variable **i** is assigned to the variable of float type f because this conversion operation is predefined in C#.

However, not all data types are implicitly converted to other data types.

- ✓ For example, **int** type cannot be converted to **uint** implicitly, it must be specify explicitly, as shown below.

Example:

```
tatic void Main(string[] args)
        {
            int i = 100;
            uint u = (uint)i;
            Console.Write(i);


        }
```

In the above example, **int** is converted to **uint** by specifying **uint** in the brackets (**uint**). This will convert an integer to **uint.**

This is called **explicit conversion**.

C# Variables

- ❖ A variable is a storage location with a type.
- ❖ Variables can have values assigned to them, and those values can be changed programmatically.
- ❖ A variable is a name given to a storage area that is used to store values of various data types.

- Each variable in C# needs to have a specific type, which determines the size and layout of the variable's memory.

For example, a variable can be of the type String, which means that it will be used to store a string value. Based on the data type, specific operations can be carried out on the variable. If we had an Integer variable, then operations such as addition and subtraction can be carried out on the variable. One can declare multiple variables in a program.

```
static public void Main()
            {

                String message = "This is EEC BE-CMP Batch ";

                //Defining a String Variable

               Int val = 2018;

               //Defining a integer variable

                Console.Write(message + val); // Displaying the values of both variable
                Console.ReadKey();
            }
```

Implicitly typed variables

- Alternatively in C#, we can declare a variable without knowing its type using ***var*** keyword. Such variables are called ***implicitly typed local variables***.
- Variables declared using ***var*** keyword must be initialized at the time of declaration.
- Var value =7;
- The compiler determines the type of variable from the value that is assigned to the variable.
- In the example var value = 7, *value* is of type *int.* This is equivalent to:
  Int value;
  Value =7;

Rules for Naming Variables in C#

There are certain rules we need to follow while naming a variable. The rules for naming a variable in C# are:

1. The variable name can contain letters (uppercase and lowercase), underscore (_) and digits only.

2. The variable name must start with either letter, underscore or @ symbol. For example,
3. C# is case sensitive. It means *age* and *Age* refers to 2 different variables.
4. A variable name must not be a C# keyword. For example, *if*, *for*, *using* can not be a variable name..

Variable Names	**Remarks**
name	Valid
subject101	Valid
_age	Valid (Best practice for naming private member variables)
@break	Valid (Used if name is a reserved keyword)
101subject	Invalid (Starts with digit)
your_name	Valid
your name	Invalid (Contains whitespace)

Best Practices for Naming a Variable

1. Choose a variable name that make sense. For example, *name*, *age*, *subject* makes more sense than *n*, *a* and *x*.

2. Use **camelCase** notation (starts with lowercase letter) for naming local variables. For example, *numberOfStudents*, *age*, etc.

3. Use **PascalCase** or **CamelCase** (starts with uppercase letter) for naming public member variables. For example, *FirstName*, *Price*, etc.

4. Use a leading underscore (_) followed by **camelCase** notation for naming private member variables. For example, *_bankBalance*, *_emailAddress*, etc.

C# Constants:

- A constant is a variable whose value cannot be changed.
- Constants are immutable values which are known at compile time and do not change for the life of the program. Constants are declared with the const modifier
- The constants are treated just like regular variables except that their values cannot be modified after their definition.
- C# does not support **const** methods, properties, or events.
- Constants must be initialized as they are declared. For example

```
class Calendar1
    {
    public const int Months = 12;
}
```

- There is no variable address associated with a constant at run time, **const** fields cannot be passed by reference and cannot appear as an l-value in an expression.
- Multiple constants of the same type can be declared at the same time, for example

```
class Calendar2
    {
        public const int Months = 12, Weeks = 52, Days = 365;
    }
```

- Constants can be marked as *public***,** *private***,** *protected***,** *internal***,** *protected internal* ***or*** *private protected***.** These access modifiers define how users of the class can access the constant

- Constants come in three flavors:
    1. **Literals**
    2. **symbolic constants**
    3. **Enumerations**.

C# Literals

- **Literal** is a value that is expressed as itself.
- For example, the number *25* or the string *"Hello World"* are both literals.

- ❖ The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called *literals.*
- ❖ Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.
- ❖ There are also enumeration constants as well.
- ❖ A **constant** is a data type that substitutes a literal.

	Constant	Literal
Example	const PI = 3.14; var radius = 5; var circumference = 2 * PI * radius;	var radius = 5; var circumference = 2 * 3.14 * radius;

Let's look at the following statement:

```
int number = 91;
```

Here,

- ✓ *int* is a data type
- ✓ *number* is a variable and
- ✓ 91 is a literal

Hence,

- ✓ Literals are fixed values that appear in the program.
- ✓ They do not require any computation.
- ✓ For example, 5, false, 'w' are literals that appear in a program directly without any computation.

Types of Literals:

I. Boolean Literals

- ✓ True and false are the available Boolean literals.
- ✓ They are used to initialize Boolean variables.

- ✓ For example: bool isValid = true;
  bool isPresent = false;

II. Integer Literals

- ✓ Integer literals are used to initialize variables of integer data types i.e. *sbyte, short, int, long, byte, ushort, uint and ulong*.
- ✓ If an integer literal ends with L or l, it is of type long. For best practice use L (not l).

```
long value1 = 4200910L;
long value2 = -10928190L;
```

- ✓ If an integer literal starts with a 0x, it represents hexadecimal value.
- ✓ Number with no prefixes are treated as decimal value.
- ✓ Octal and binary representation are not allowed in C#.

```
int decimalValue = 25;
int hexValue = 0x11c;// decimal value 284
```

III. Floating Point Literals

- ✓ Floating point literals are used to initialize variables of float and double data types.
- ✓ If a floating point literal ends with a suffix f or F, it is of type float. Similarly, if it ends with d or D, it is of type double. If neither of the suffix is present, it is of type double by **default**.

IV. Character and String Literals

- ✓ Character literals are used to initialize variables of char data types.
- ✓ Character literals are enclosed in single quotes. For example, 'x', 'p', etc.
- ✓ String literals are the collection of character literals.
- ✓ They are enclosed in double quotes. For example, "Hello", ”Programming Technology", etc.

```
string firstName = "Programming";
string lastName = " Technology";
```

Symbolic constants:

- ❖ Symbolic constants assign a name to a constant value.
- ❖ You declare a symbolic constant using the **const** keyword and the following syntax: const type identifier = value;

- A constant must be initialized when it is declared, and once initialized it cannot be altered. For example: const int FreezingPoint = 32;

In this declaration, 32 is a literal constant and FreezingPoint is a symbolic constant of type int.

Example illustrates the use of symbolic constants.

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

                const int FreezingPoint = 32;   // degrees Farenheit
                const int BoilingPoint = 212;

           System.Console.WriteLine("Freezing point of water: {0}",FreezingPoint);
           System.Console.WriteLine("Boiling point of water: {0}",BoilingPoint);
            Console.ReadLine();


            }
```

Enumerations

- Enumerations provide a powerful alternative to constants.
- An enumeration is a distinct value type, consisting of a set of named constants (called the enumerator list).
- Every enumeration has an underlying type, which can be any integral type (integer, short, long, etc.) except for char.
- The technical definition of an enumeration is:

[attributes][modifiers] enum identifier [: base-type] { enumerator-list};

- An enumeration begins with the keyword *enum*, which is generally followed by an identifier, such as: enum Days

Example: enum in C#

```
class EnumExample
    {
        enum Days
              {
            Sun,
            Mon,
            tue,
            Wed,
            thu,
            Fri,
            Sat
              };

        static void Main(string[] args)
        {
            Console.Write(Days.Sun);

            Console.ReadKey();
        }
    }
```

- An **enum** declaration ends with the enumerator list. The enumerator list contains the constant assignments for the enumeration, each separated by a comma (,).

C# Identifier:

- Identifiers are the name given to entities such as variables, methods, classes, etc.
- An **identifier** is the name you assign to a type (class, interface, struct, delegate, or enum), member, variable, or namespace.
- In programming languages, identifiers are used for identification purpose. Or in other words, identifiers are the user-defined name of the program components.
- In C#, an identifier can be a class name, method name, variable name or a label.
- They are tokens in a program which uniquely identify an element. For example, int value;
- Here, *value* is the name of variable. Hence it is an identifier. Reserved keywords cannot be used as identifiers unless @ is added as prefix. For example, int break;
    - This statement will generate an error in compile time.
- Identifiers in C# are case sensitive.

- By convention, C# programs use PascalCase for type names, namespaces, and all public members
- Example:

```
public class EEC
        {
            static public void Main()
            {
                int x;
            }
        }
```

Here the total number of identifers present in the above example is 3 and the names of these identifiers are:

- ✓ **EEC:** Name of the class
- ✓ **Main:** Method name
- ✓ **x:** Variable name

```
class EEC
        {

            // Main Method
            static public void Main()
            {

                // variable
                int a = 10;
                int b = 39;
                int c;

                // simple addition
                c = a + b;
                Console.WriteLine("The sum of two number is: {0}", c);
            }
        }
```

Output:

The sum of two number is: 49

C# Keywords

Keywords are predefined sets of reserved words that have special meaning in a program. The meaning of keywords cannot be changed, neither can they be directly used as identifiers in a program.

For example, long mobileNum;

Here,

- ✓ *long* is a keyword and *mobileNum* is a variable (identifier).
- ✓ long has a special meaning in C# i.e. it is used to declare variables of type long and this function cannot be changed.
- ✓ Also, keywords like long, int, char, etc cannot be used as identifiers. So, we cannot have something like: long long;

- ❖ C# has a total of 79 keywords. All these keywords are in lowercase. Here is a complete list of all C# 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	in (generic modifier)	int	interface

internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
Sealed	short	sizeof	stackalloc
Static	string	struct	switch
This	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	using static
Void	volatile	while	

- ✓ Although keywords are reserved words, they can be used as identifiers if @ is added as prefix. For example, `int @void;`
- ✓ The above statement will create a variable *@void* of type int.

Contextual Keywords

Besides regular keywords, C# has 25 contextual keywords. Contextual keywords have specific meaning in a limited program context and can be used as identifiers outside that context. They are not reserved words in C#.

Add	alias	ascending
Async	await	descending
dynamic	from	get
global	group	into
Join	let	orderby
partial (type)	partial (method)	remove
Select	set	value
Var	when (filter condition)	where (generic type constraint)
Yield		