Principle of Programing

2. Data type

Lecturers:

Sipan M. Hameed

www.sipan.dev

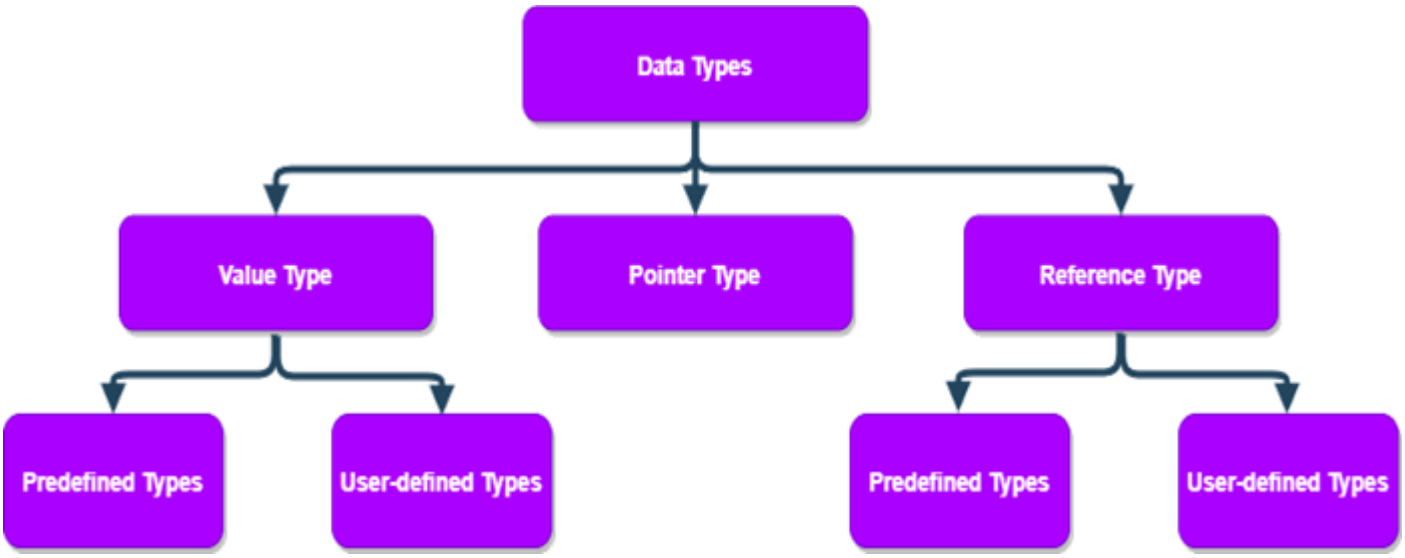
2024 - 2025

1. Table of Contents

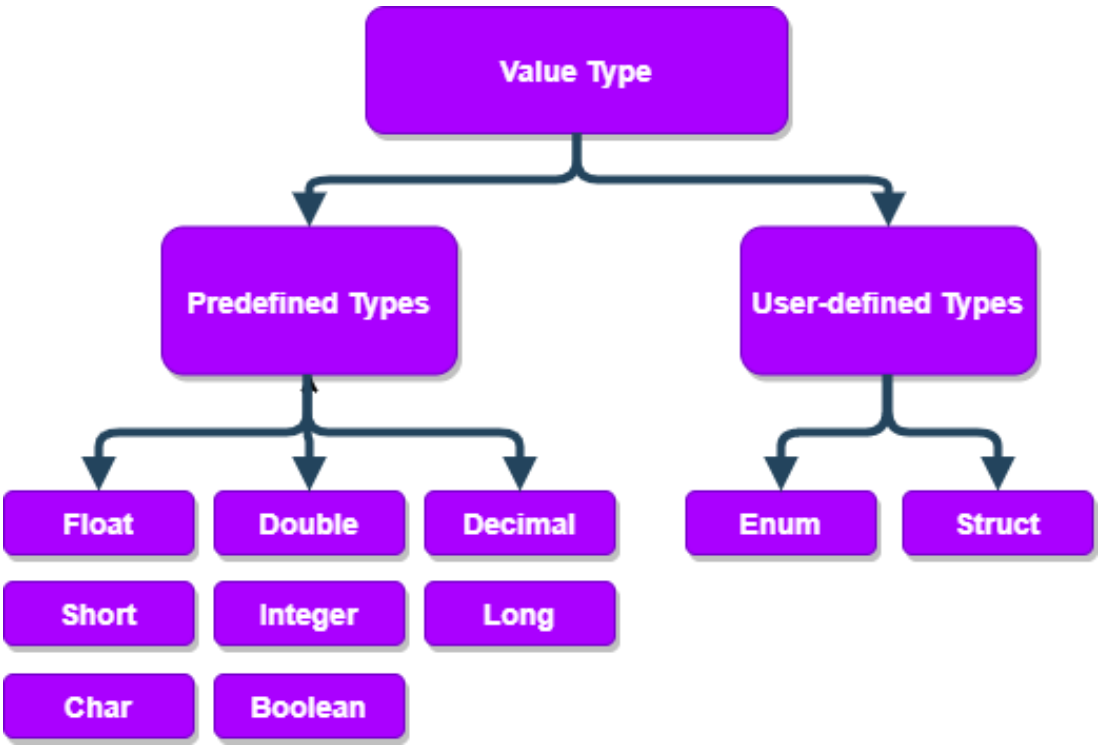
2. Data type

In computer science and computer programming, a data type or simply type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

C# has 3 categories about variable types, as shown below:



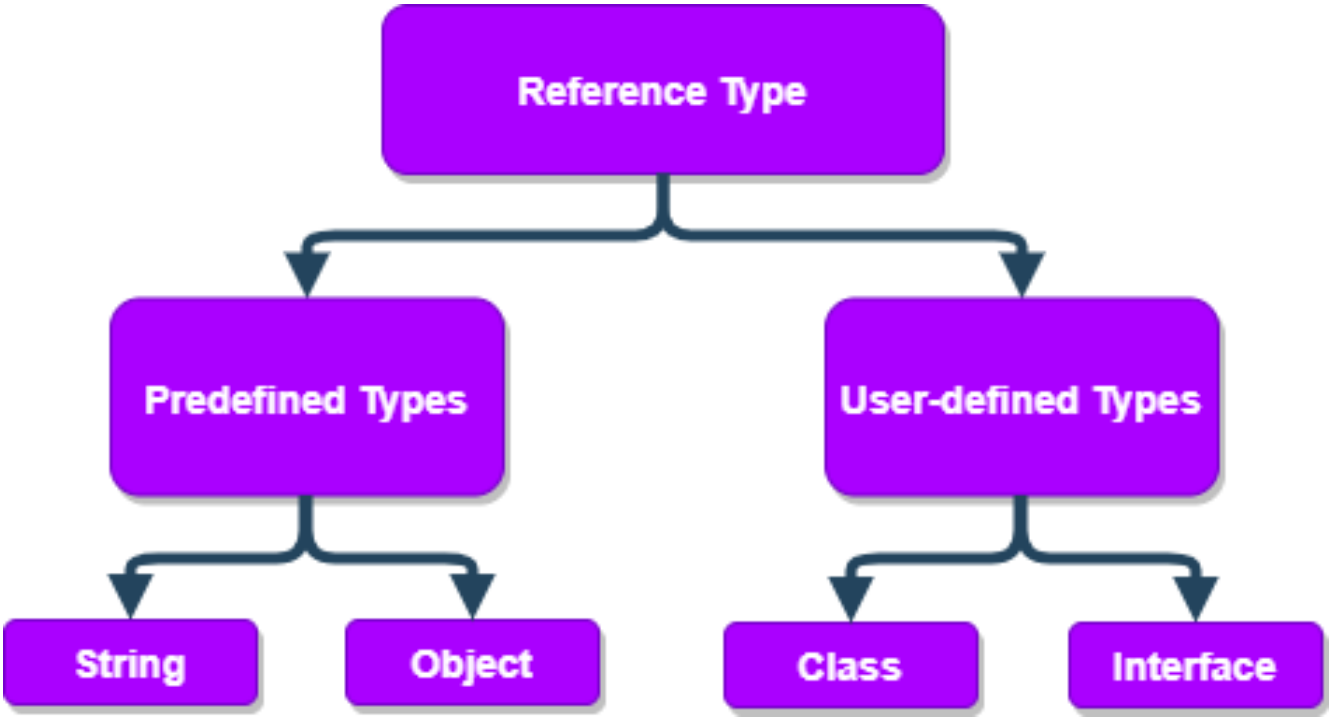
2.1 Value Type



Data Type	Represents	Memory Size	Range	Default Value
Bool	Boolean Value	1 byte	True or False	False
Byte	Unsigned Integer	1 byte	0 to 255	0
Char	Unicode character	1 byte	U +0000 to U +ffff	'\0'
Short	Signed integer type	2 byte	-32,768 to 32,767	0
signed short	Signed integer type	2 byte	-32,768 to 32,767	0
unsigned short	Unsigned integer type	2 byte	0 to 65,535	0
int	Signed integer type	4 byte	-2,147,483,648 to 2,147,483,647	0
signed int	Signed integer type	4 byte	-2,147,483,648 to 2,147,483,647	0
unsigned int	Unsigned integer type	4 byte	0 to 4,294,967,295	0
long	Signed integer type	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
signed long	Signed integer type	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
unsigned long	Unsigned integer type	8 byte	0 to 18,446,744,073,709,551,615	0
float	Single precision floating point type	4 byte	-3.4 x 1038 to + 3.4 x 1038	0.0F
double	Double-precision floating point type	8 byte	(+/-)5.0 x 10-324 to (+/-)1.7 x 10308	0.0D
decimal	Decimal values with 28-29 significant digits	16 byte	(-7.9 x 1028 to 7.9 x 1028) / 100 to 28	0.0M

2.2 Reference Type:

.



2.3 Pointer Type:

Pointer type keeps the memory address of arrays and value types. Garbage collector does not track pointer types. Unsafe context is used to run code outside the control of Garbage Collector.

2.4 C# Program structure

The key organizational concepts in C# are ***programs***, ***namespaces***, ***types***, ***members***, and ***assemblies***. A typical C# Program consists of several different parts as shown below:

1. Namespace
2. Class
3. The main method
4. Methods inside the class
5. Class definition or class attributes
6. Blocks
7. Expressions
8. Statements
9. Comments

2.4.1 Basic Structure of C# Program

2.4.1.1 visual studio

Visual Studio is one of the Microsoft IDE tools. Using this tool, we can implement applications with .NET framework. This tool provides some features such as

1. Editor
2. Compiler
3. Interpreter

2.4.1.2 What is a console application?

1. These applications contain a similar user interface to the OS like MS-DOS, UNIX, etc.
2. The console application is known as CUI application because in this application we completely work with CUI environment.
3. These applications are similar to c or c++ applications.
4. Console applications do not provide any GUI facilities like the mouse pointer, colors, buttons, menu bars, etc.

2.4.1.3 What is a console application?

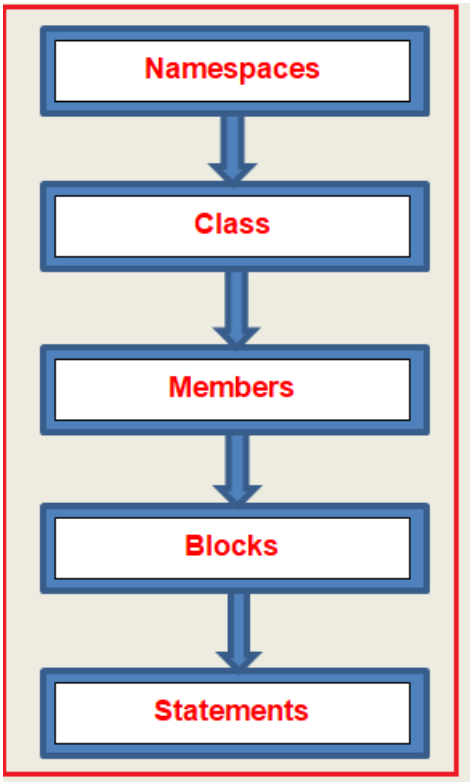
1. These applications contain a similar user interface to the OS like MS-DOS, UNIX, etc.
2. The console application is known as CUI application because in this application we completely work with CUI environment.
3. These applications are similar to c or c++ applications.

4. Console applications do not provide any GUI facilities like the mouse pointer, colors, buttons, menu bars, etc.

Let's First Understand the Basic Structure of C# Program using a console application.

```
//List of classlibraries /namespaces
namespace namespacename
{
    class classname
    {
        static void Main(string[] args)
        {
            //statements
        }
    }
}
```

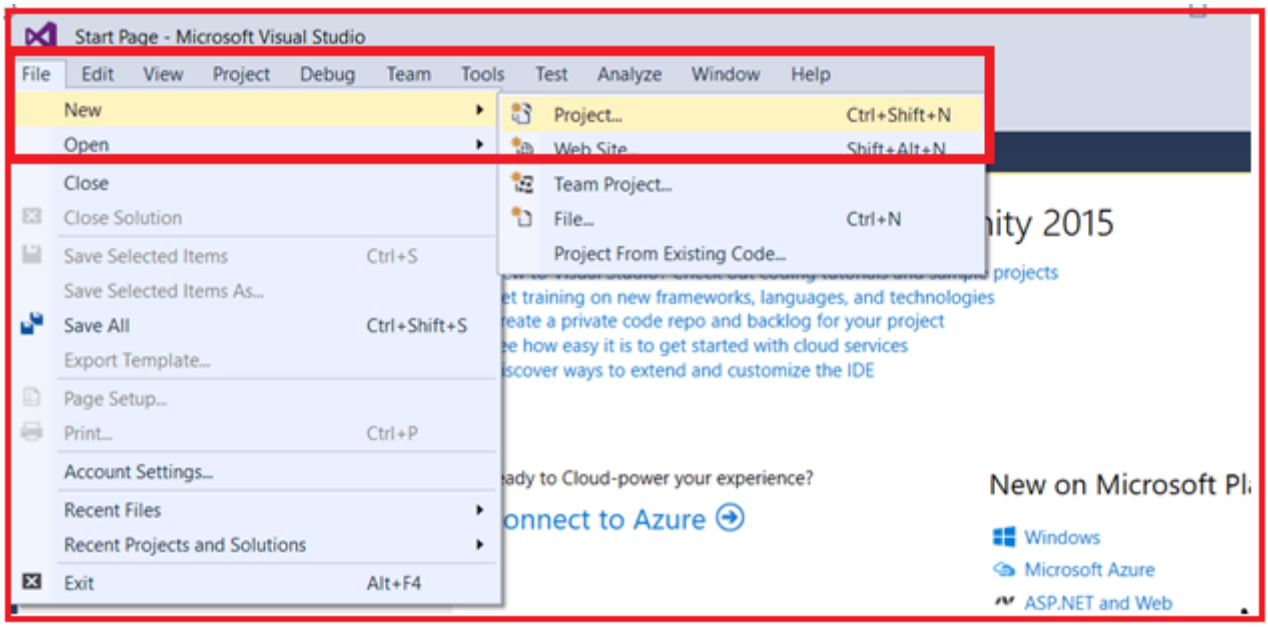
The above process is shown in the below diagram.



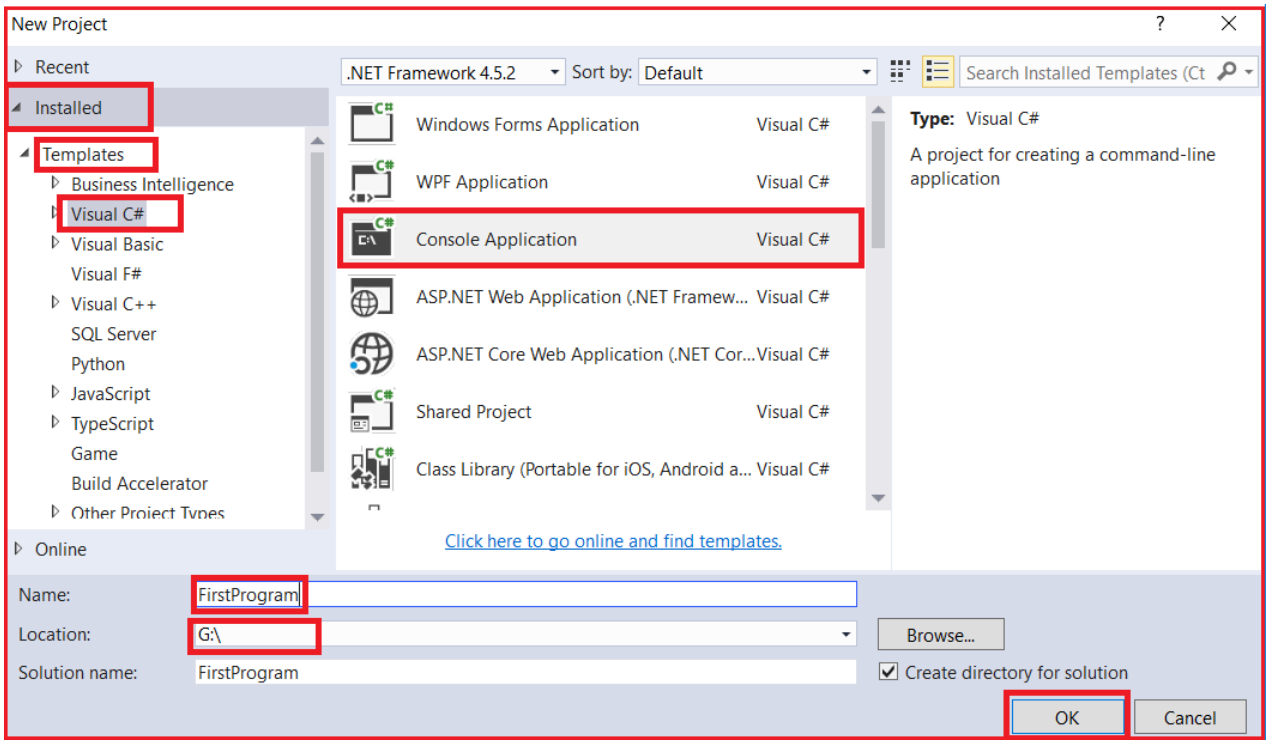
NOTE: C#.NET is a case-sensitive language and every statement in C# should end with a semicolon.

2.4.1.4 How to Create a console application using the visual studio?

Open visual studio 2015 (You can use any lower or higher version). Go to the new project (**File => New => Project**) as shown in the below image.



In the next window, select **Installed => Templates => Visual C#** from the left pane. From the middle Panel, select **Console Application**. Then specify the application name and location and finally click on the **OK** button as shown in the below image.



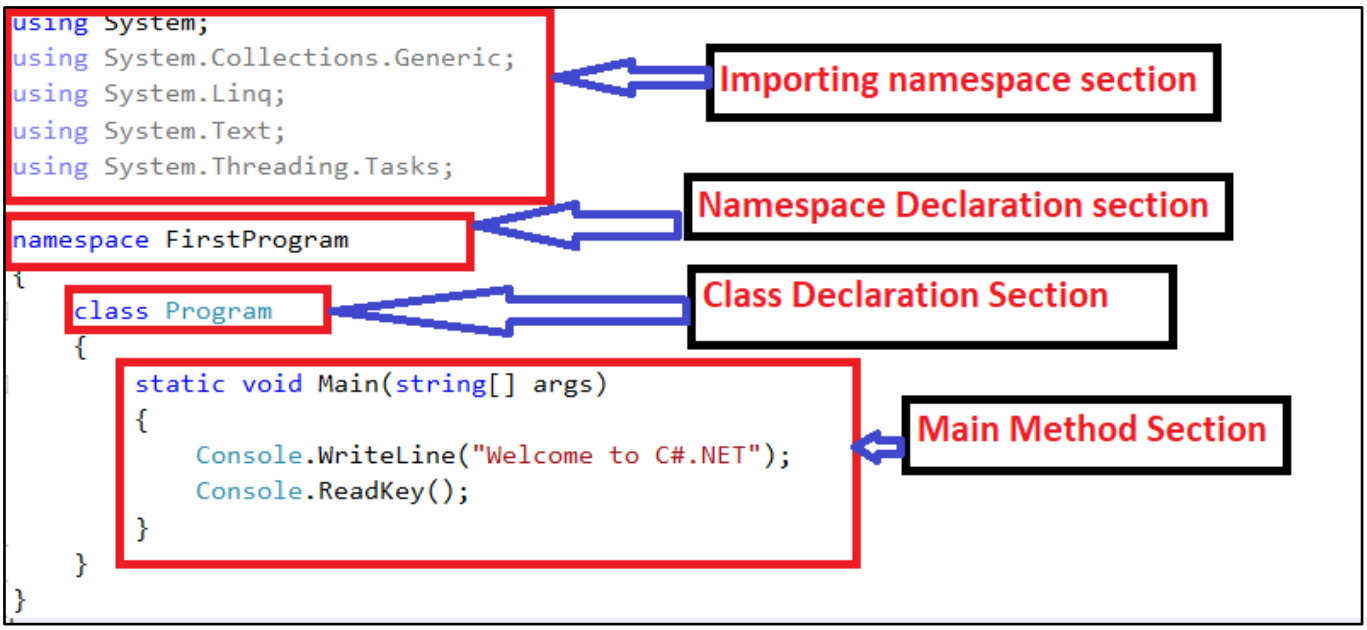
Once you click on the OK button, it will take some time to create the project solution for us.

2.4.1.5 Let's create one program to print a welcome message on the console.

Whenever you create a console application, by default the .NET Framework creates a class (i.e. Program class) for us. Let's modify the Program.cs file as shown below to print a welcome message on the console screen.

```
using System;
namespace FirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C#.NET");
            Console.ReadKey();
        }
    }
}
```

Through visual studio whenever we are creating one console application, automatically we are getting four sections as shown in the image below.



Let's understand each of these sections in details.

2.4.1.6 Importing section:

This section contains importing statements that are used to import the BCL (Base Class Libraries). This is similar to the include statements in the C programming language.

Syntax: **using namespace;**
Example: **using System;**

Note: If the required namespace is a member of another namespace we have to specify the parent and child namespaces separated by a dot.

```
using System.Data;
using System.IO;
```

2.4.1.7 Namespace Declaration:

Here a user-defined namespace is to be declared. In .NET applications, all classes related to the project should be declared inside one namespace.

Syntax:

```
namespace NamespaceName
{
}
```

Generally, the namespace name will be the same as the project name.

2.4.1.8 Class Declaration:

This is to declare the start-up class of the project. In every .NET applications like console and windows applications, there should be a start-up class. In this application, the start-up class name is "program". A startup class is nothing but a class which contains a "Main()" method.

Syntax:

```
class ClassName
{
}
```

2.4.1.9 Main() method:

The main() method is the starting execution point of the application. When the application is executed the main method will be executed first. This method contains the main logic of the application.

2.4.1.10 What is using?

Using is a keyword. Using this keyword, we can refer to .NET in C# applications.

2.4.2 C# Keywords

Keywords are predefined, reserved identifiers that have special meanings to the compiler. They cannot be used as identifiers in your program unless they include `@`.

The first table in this topic lists keywords that are reserved identifiers in any part of a C# program. The second table in this topic lists the contextual keywords in C#. Contextual keywords have special meaning only in a limited program context and can be used as identifiers outside that context.

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	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	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	virtual	void	volatile
while			

2.5 Summery Data Types

Data Type	Represents	Memory Size	Range	Default Value
Bool	Boolean Value	1 byte	True or False	False
Char	Unicode character	1 byte	U +0000 to U +ffff	'\0'
int	Signed integer type	4 byte	-2,147,483,648 to 2,147,483,647	0
double	Double-precision floating point type	8 byte	(+/-)5.0 x 10-324 to (+/-)1.7 x 10308	0.0D
string	Stores a sequence of characters, surrounded by double quotes	2 bytes per character		null

2.5.1 C# Data Types

As explained in the variables chapter, a variable in C# must be a specified data type:

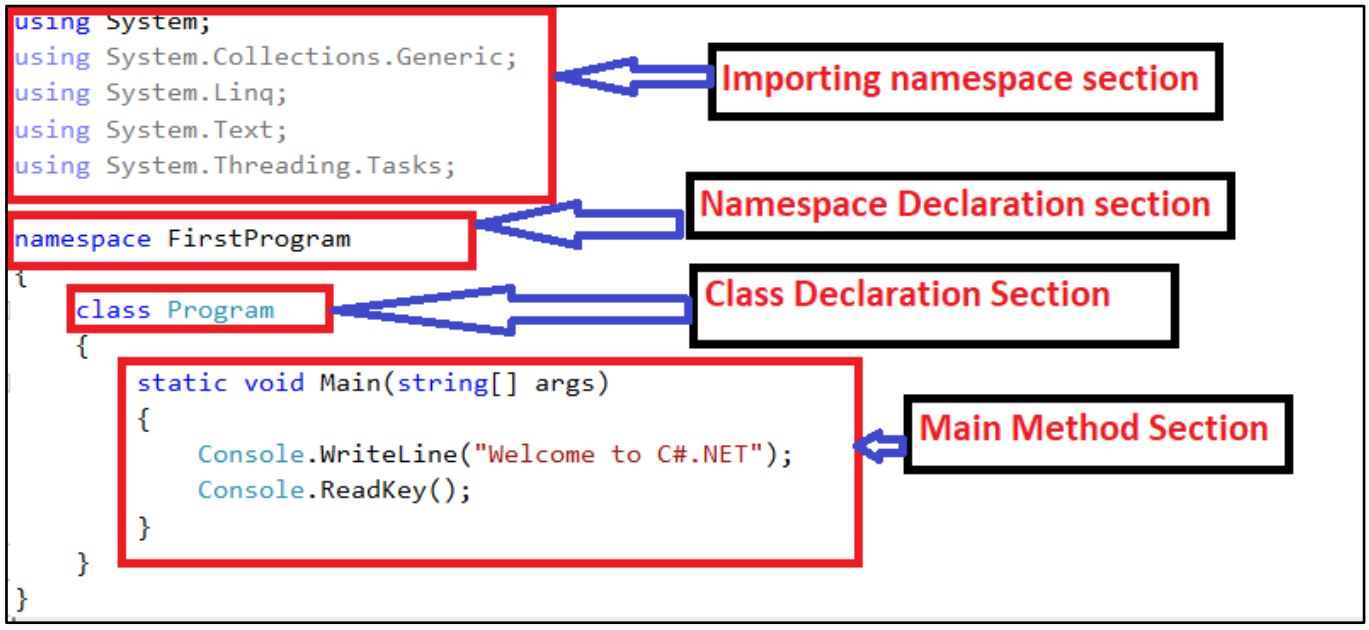
Example

```
int myNum = 5;               // Integer (whole number)
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

2.5.2 Declaration namespace, class and method

```
using System;
namespace FirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C#.NET");
            Console.ReadKey();
        }
    }
}
```

Through visual studio whenever we are creating one console application, automatically we are getting four sections as shown in the image below.



2.5.2.1 Namespace Declaration:

Syntax:

```
namespace Firstprogram
{



}
```

2.5.2.2 Class Declaration:

Syntax:

```
class program
{

}
```

2.5.2.3 method Declaration (function Declaration)

Syntax:

```
Static void Main(string[] arg)
{

}
```

2.5.3 C# Variables

Variables are containers for storing data values.

In C#, there are different types of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- double - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

2.5.4 Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Where type is a C# type (such as int or string), and variableName is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called name of type string and assign it the value "John":

```
string name = "John";
Console.WriteLine(name);
```

To create a variable that should store a number, look at the following example:

Example: Create a variable called myNum of type int and assign it the value 15:

```
int myNum = 15;
Console.WriteLine(myNum);
```

You can also declare a variable without assigning the value, and assign the value later:

Example:

```
int myNum;
myNum = 15;
Console.WriteLine(myNum);
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of myNum to 20:

```
int myNum = 15;
myNum = 20; // myNum is now 20
Console.WriteLine(myNum);
```

2.5.5 Constants

However, you can add the const keyword if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "constant", which means unchangeable and read-only):

Example

```
const int myNum = 15;
myNum = 20; // error
```

2.5.6 Display Variables

The WriteLine() method is often used to display variable values to the console window.

To combine both text and a variable, use the + character:

Example

```
string name = "John";
Console.WriteLine("Hello " + name);
```

You can also use the + character to add a variable to another variable:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
Console.WriteLine(fullName);
```

2.5.7 Declare Many Variables

To declare more than one variable of the same type, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
Console.WriteLine(x + y + z);
```

2.5.8 C# Identifiers

All C# variables must be identified with unique names.These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

2.5.9 general rules for constructing names

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits and the underscore character (_)
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace

- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like C# keywords, such as int or double) cannot be used as names

2.5.10 C# Comments

Comments can be used to explain C# code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

2.5.10.1 Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by C# (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment
Console.WriteLine("Hello World!");
```

This example uses a single-line comment at the end of a line of code:

Example

```
Console.WriteLine("Hello World!");  // This is a comment
```

2.5.10.2 C# Multi-line Comments

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by C#.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
Console.WriteLine("Hello World!");
```

2.5.11 Get User Input

You have already learned that **Console.WriteLine()** is used to output (print) values. Now we will use **Console.ReadLine()** to get user input.

In the following example, the user can input his or hers username, which is stored in the variable **userName**. Then we print the value of **userName**:

Example

```
// Type your username and press enter
Console.WriteLine("Enter username:");

// Create a string variable and get user input from the keyboard and
store it in the variable
string userName = Console.ReadLine();

// Print the value of the variable (userName), which will display the
input value
Console.WriteLine("Username is: " + userName);
```

2.5.11.1 User Input and Numbers

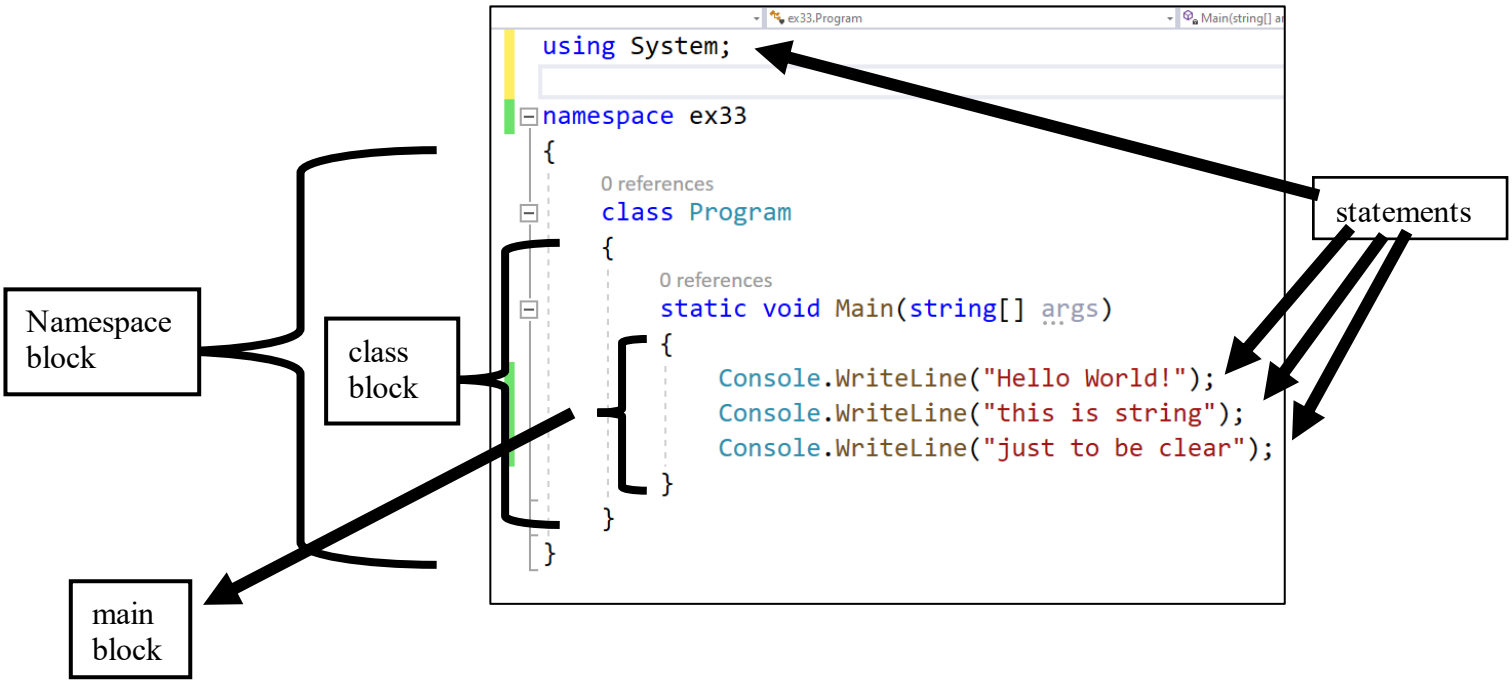
Luckily, for you, you just learned from the previous chapter (Type Casting), that you can convert any type explicitly, by using one of the **Convert.To** methods:

Example

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

2.6 Practical examples

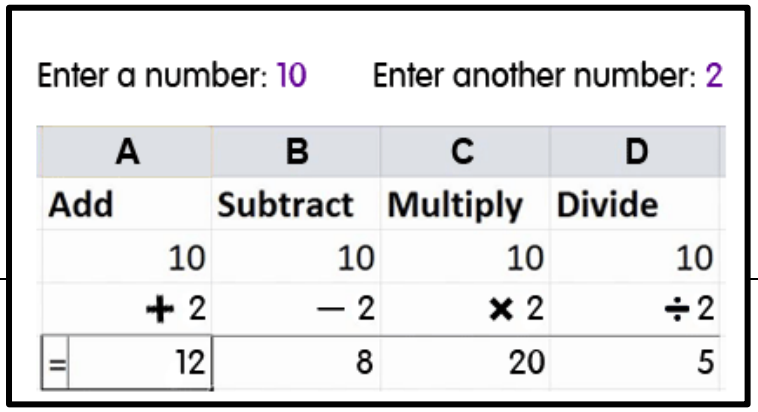
2.6.1 C#: example: Block and statement




```
using System;

namespace ex33
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("this is string");
            Console.WriteLine("just to be clear");
        }
    }
}
```

2.6.2 C# Exercises-7

Write a C# Sharp program to print on screen the output of adding, subtracting, multiplying and dividing of two numbers which will be entered by the user.




```
using System;
namespace ex7
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a number: ");
            int num1 = Convert.ToInt32(Console.ReadLine());

            Console.Write("Enter another number: ");
            int num2 = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("{0} + {1} = {2}", num1, num2, num1 + num2);
            Console.WriteLine("{0} - {1} = {2}", num1, num2, num1 - num2);
            Console.WriteLine("{0} x {1} = {2}", num1, num2, num1 * num2);
            Console.WriteLine("{0} / {1} = {2}", num1, num2, num1 / num2);
            Console.WriteLine("{0} mod {1} = {2}", num1, num2, num1 % num2);

        }
    }
}
```

Output:

```
Enter a number: 10
Enter another number: 2
10 + 2 = 12
10 - 2 = 8
10 x 2 = 20
10 / 2 = 5
10 mod 2 = 0
```

2.6.3 C# Sharp-8: Print the average of four numbers

C# Sharp Code:

```
using System;

namespace ex8
{
    class Program
    {
        static void Main(string[] args)
        {
            double number1, number2, number3, number4;

            Console.Write("Enter the First number: ");
            number1 = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter the Second number: ");
            number2 = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter the third number: ");
            number3 = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter the fourth number: ");
            number4 = Convert.ToDouble(Console.ReadLine());

            double result = (number1 + number2 + number3 + number4) / 4;
            Console.WriteLine("The average of {0}, {1}, {2}, {3} is: {4} ",
         number1, number2, number3, number4, result);

        }
    }
}
```

Output:

```
Enter the First number: 17
Enter the Second number: 17
Enter the third number: 517
Enter the four number: 51
The average of 17, 17, 517, 51 is: 150.5
```

2.6.4 C# Exercises-9:

Program to take a number as input and print its multiplication table

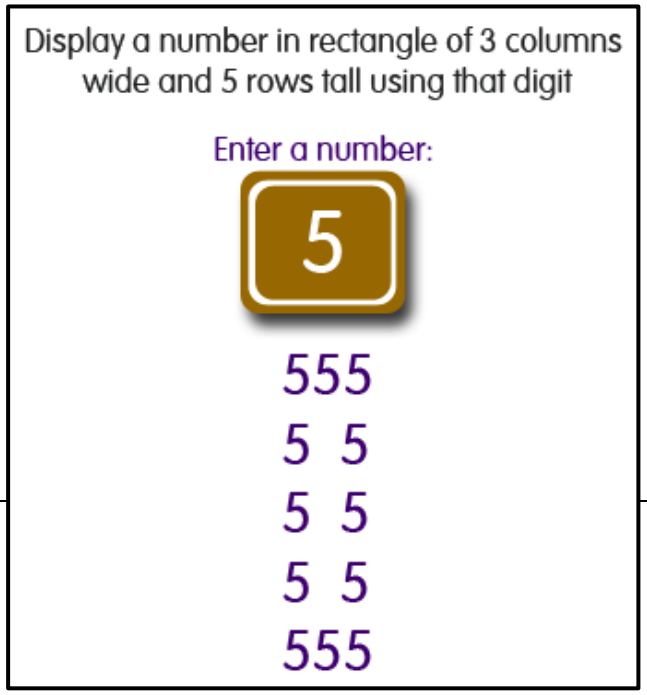
```
using System;

namespace ex9
{
    class Program
    {
        static void Main(string[] args)
        {
            int x;
            int result;

            Console.WriteLine("Enter a number:");
            x = Convert.ToInt32(Console.ReadLine());

            result = x * 1;
            Console.WriteLine("The table is : {0} x {1} = {2}", x, 1, result);
            result = x * 2;
            Console.WriteLine("             : {0} x {1} = {2}", x, 2, result);
            result = x * 3;
            Console.WriteLine("             : {0} x {1} = {2}", x, 3, result);
            result = x * 4;
            Console.WriteLine("             : {0} x {1} = {2}", x, 4, result);
            result = x * 5;
            Console.WriteLine("             : {0} x {1} = {2}", x, 5, result);
            result = x * 6;
            Console.WriteLine("             : {0} x {1} = {2}", x, 6, result);
            result = x * 7;
            Console.WriteLine("             : {0} x {1} = {2}", x, 7, result);
            result = x * 8;
            Console.WriteLine("             : {0} x {1} = {2}", x, 8, result);
            result = x * 9;
            Console.WriteLine("             : {0} x {1} = {2}", x, 9, result);
            result = x * 10;
            Console.WriteLine("             : {0} x {1} = {2}", x, 10, result);

        }
    }
}
```

Output:

```
Enter a number:
5
The table is : 5 x 1 = 5
             : 5 x 2 = 10
             : 5 x 3 = 15
             : 5 x 4 = 20
             : 5 x 5 = 25
             : 5 x 6 = 30
             : 5 x 7 = 35
             : 5 x 8 = 40
             : 5 x 9 = 45
             : 5 x 10 = 50
```

2.6.5 C# Exercises-10

Display a number in rectangle of 3 columns wide and 5 rows tall using that digit



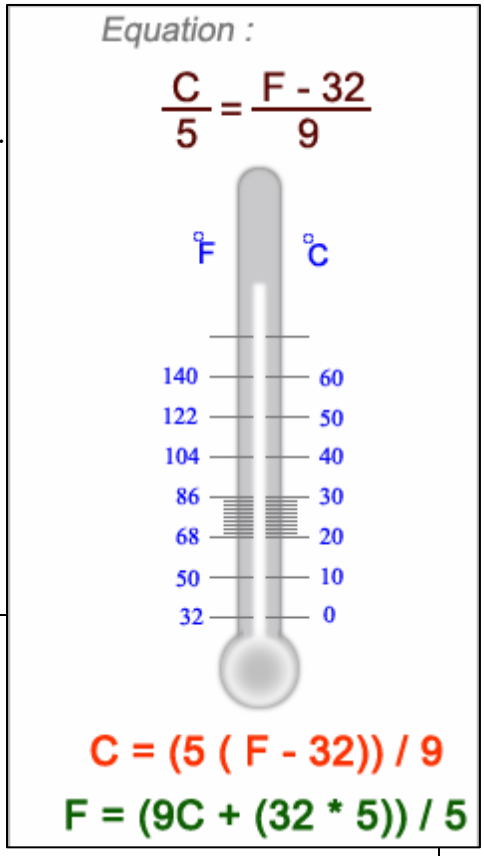

```
using System;

namespace ex10
{
    class Program
    {
        static void Main(string[] args)
        {
            int x;
            Console.Write("Enter a number: ");
            x = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("{0}{0}{0}", x);
            Console.WriteLine("{0} {0}", x);
            Console.WriteLine("{0} {0}", x);
            Console.WriteLine("{0} {0}", x);
            Console.WriteLine("{0}{0}{0}", x);

        }
    }
}
```

Output:

```
Enter a number: 5
555
5 5
5 5
5 5
555
```

2.6.6 C# Exercises-11

Program to convert from Celsius degrees to Kelvin and Fahrenheit.



kelvin = celsius + 273
fahrenheit = celsius x 18 / 10 + 32

C# Sharp Code:

```
using System;
public class Exercise14
{
   public static void Main( )
    {
        Console.Write("Enter the amount of celsius: ");
        int celsius = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Kelvin = {0}", celsius + 273);
        Console.WriteLine("Fahrenheit = {0}", celsius * 18 / 10 + 32);
    }
}
```

Output:

```
Enter the amount of celsius: 40
Kelvin = 313
Fahrenheit = 104
```