The “Hello World!” program is often the first program we see when we dive into a new language. It simply prints **Hello World!** on the output screen.

The purpose of this program is to get us familiar with the basic syntax and requirements of a programming language.

**"Hello World!" in C#**

// Hello World! program

namespace HelloWorld

{

class Hello {

static void Main(string[] args)

{

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

}

}

}

When you run the program, the output will be:

Hello World!

**How the "Hello World!" program in C# works?**

Let's break down the program line by line.

1. // Hello World! Program  
     
   // indicates the beginning of a comment in C#. Comments are not executed by the C# compiler.  
     
   They are intended for the developers to better understand a piece of code. To learn more about comments in C#, visit *C# comments*.
2. namespace HelloWorld{...}  
     
   The namespace keyword is used to define our own namespace. Here we are creating a namespace called HelloWorld.  
     
   Just think of namespace as a container which consists of classes, methods and other namespaces. To get a detailed overview of namespaces, *visit*[C# Namespaces](https://www.programiz.com/csharp-programming/namespaces).
3. class Hello{...}  
     
   The above statement creates a class named - Hello in C#. Since, C# is an object-oriented programming language, creating a class is mandatory for the program’s execution.
4. static void Main(string[] args){...}  
     
   Main() is a method of class Hello. The execution of every C# program starts from the Main() method. So it is mandatory for a C# program to have a Main() method.  
     
   The signature/syntax of the Main() method is:
5. static void Main(string[] args)
6. {
7. ...

}

We’ll learn more about methods in the later chapters.

1. System.Console.WriteLine("Hello World!");  
     
   For now, just remember that this is the piece of code that prints **Hello World!** to the output screen.You’ll learn more about how it works in the later chapters.

**Alternative Hello World! implementation**

Here’s an alternative way to write the “Hello World!” program.

// Hello World! program

using System;

namespace HelloWorld

{

class Hello {

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

# C# Command Line Arguments

Arguments that are passed by command line known as command line arguments. We can send arguments to the Main method while executing the code. The string **args** variable contains all the values passed from the command line.

In the following example, we are passing command line arguments during execution of program.

1. **using** System;
2. **namespace** CSharpProgram
3. {
4. **class** Program
5. {
6. // Main function, execution entry point of the program
7. **static** **void** Main(**string**[] args) // string type parameters
8. {
9. // Command line arguments
10. Console.WriteLine("Argument length: "+args.Length);
11. Console.WriteLine("Supplied Arguments are:");
12. **foreach** (Object obj **in** args)
13. {
14. Console.WriteLine(obj);
15. }
16. }
17. }
18. }

Compile and execute this program by using following commands.

**Compile:** csc Program.cs

**Execute:** Program.exe Hi there, how are you?

After executing the code, it produces the following output to the console.

**Output:**

Argument length: 5

Supplied Arguments are:

Hi

there,

how

are

you?

### Difference between WriteLine() and Write() method

The main difference between WriteLine() and Write() is that the Write() method only prints the string provided to it, while the WriteLine() method prints the string and moves to the start of next line as well.

Let's take at a look at the example below to understand the difference between these methods.

#### How to use WriteLine() and Write() method?

using System;

namespace Sample

{

class Test

{

public static void Main(string[] args)

{

Console.WriteLine("Prints on ");

Console.WriteLine("New line");

Console.Write("Prints on ");

Console.Write("Same line");

}

}

}

When we run the program, the output will be

Prints on

New line

Prints on Same line

### Printing Variables and Literals using WriteLine() and Write()

The WriteLine() and Write() method can be used to print variables and literals. Here's an example.

#### Printing Variables and Literals

using System;

namespace Sample

{

class Test

{

public static void Main(string[] args)

{

int value = 10;

// Variable

Console.WriteLine(value);

// Literal

Console.WriteLine(50.05);

}

}

}

When we run the program, the output will be

10

50.05

### Combining (Concatenating) two strings using + operator and printing them

Strings can be combined/concatenated using the + operator while printing.

#### Printing Concatenated String using + operator

using System;

namespace Sample

{

class Test

{

public static void Main(string[] args)

{

int val = 55;

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

Console.WriteLine("Value = " + val);

}

}

}

When we run the program, the output will be

Hello World

Value = 55

### Printing concatenated string using Formatted String [Better Alternative]

A better alternative for printing concatenated string is using formatted string. Formatted string allows programmer to use placeholders for variables. For example,

The following line,

Console.WriteLine("Value = " + val);

can be replaced by,

Console.WriteLine("Value = {0}", val);

{0} is the placeholder for variable val which will be replaced by value of val. Since only one variable is used so there is only one placeholder.

Multiple variables can be used in the formatted string. We will see that in the example below.

#### Printing Concatenated string using String formatting

using System;

namespace Sample

{

class Test

{

public static void Main(string[] args)

{

int firstNumber = 5, secondNumber = 10, result;

result = firstNumber + secondNumber;

Console.WriteLine("{0} + {1} = {2}", firstNumber, secondNumber, result);

}

}

}

When we run the program, the output will be

5 + 10 = 15

Here, {0} is replaced by firstNumber, {1} is replaced by secondNumber and {2} is replaced by result. This approach of printing output is more readable and less error prone than using + operator.

## 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.

### Get String Input From User

using System;

namespace Sample

{

class Test

{

public static void Main(string[] args)

{

string testString;

Console.Write("Enter a string - ");

testString = Console.ReadLine();

Console.WriteLine("You entered '{0}'", testString);

}

}

}

When we run the program, the output will be:

Enter a string - Hello World

You entered 'Hello World'

### Difference between ReadLine(), Read() and ReadKey() method:

The difference between ReadLine(), Read() and ReadKey() method is:

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

#### Difference between Read() and ReadKey() method

using System;

namespace Sample

{

class Test

{

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);

}

}

}

When we run the program, the output will be

Press any key to continue...

x

Input using Read() - Learning C#

Ascii Value = 76

From this example, it must be clear how ReadKey() and Read() method works. While using ReadKey(), as soon as the key is pressed, it is displayed on the screen.

When Read() is used, it takes a whole line but only returns the ASCII value of first character. Hence, 76 (ASCII value of L) is printed.

### Reading numeric values (integer and floating point types)

Reading a character or string is very simple in C#. All you need to do is call the corresponding methods as required.

But, reading numeric values can be slightly tricky in C#. We’ll still use the same ReadLine()method we used for getting string values. But since the ReadLine() method receives the input as string, it needs to be converted into integer or floating point type.

One simple approach for converting our input is using the methods of Convert class.

#### Reading Numeric Values from User using Convert class

using System;

namespace UserInput

{

class MyClass

{

public static void Main(string[] args)

{

string userInput;

int intVal;

double doubleVal;

Console.Write("Enter integer value: ");

userInput = Console.ReadLine();

/\* Converts to integer type \*/

intVal = Convert.ToInt32(userInput);

Console.WriteLine("You entered {0}",intVal);

Console.Write("Enter double value: ");

userInput = Console.ReadLine();

/\* Converts to double type \*/

doubleVal = Convert.ToDouble(userInput);

Console.WriteLine("You entered {0}",doubleVal);

}

}

}

When we run the program, the output will be

Enter integer value: 101

You entered 101

Enter double value: 59.412

You entered 59.412