**Q1: How to write Hello program in C# and explain in detail?**

**A: Code:**

using System;

namespace HelloWorldApplication {

class HelloWorld {

static void Main(string[] args) {

/\* my first program in C# \*/

Console.WriteLine("Hello World");

Console.ReadKey();

}

}

}

When this code is compiled and executed, it produces the following result −

“Hello World”

* The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.
* The next line has the **namespace** declaration. A **namespace** is a collection of classes. The *HelloWorldApplication* namespace contains the class *HelloWorld*.
* The next line has a **class** declaration, the class *HelloWorld* contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the *HelloWorld* class has only one method **Main**.
* The next line defines the **Main** method, which is the **entry point** for all C# programs. The **Main** method states what the class does when executed.
* The next line /\*...\*/ is ignored by the compiler and it is put to add **comments** in the program.
* The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**

*WriteLine* is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

The last line **Console.ReadKey();** is for the VS.NET Users. This makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

**Compilation And Execution:**

If you are using Visual Studio.Net for compiling and executing C# programs, take the following steps −

* Start Visual Studio.
* On the menu bar, choose File -> New -> Project.
* Choose Visual C# from templates, and then choose Windows.
* Choose Console Application.
* Specify a name for your project and click OK button.
* This creates a new project in Solution Explorer.
* Write code in the Code Editor.
* Click the Run button or press F5 key to execute the project. A Command Prompt window appears that contains the line Hello World.

You can compile a C# program by using the command-line instead of the Visual Studio IDE −

* Open a text editor and add the above-mentioned code.
* Save the file as **helloworld.cs**
* Open the command prompt tool and go to the directory where you saved the file.
* Type **csc helloworld.cs** and press enter to compile your code.
* If there are no errors in your code, the command prompt takes you to the next line and generates **helloworld.exe** executable file.
* Type **helloworld** to execute your program.
* You can see the output Hello World printed on the screen.

**Q2: (a) Write a simple C# program taking two different strings in text boxes, concatenate them and display with the help of message box and explain in detail.**

**A:** **Code:**

using System;

public class C {

static public void Main()

{

string strA = "Hello! ";

string strB = "World.";

string str;

// print all strings

Console.WriteLine("String A is: {0}", strA);

Console.WriteLine("String B is: {0}", strB);

// Concatenate two different strings

// into a single String

// using Concat(String, String ) Method

str = String.Concat(strA, strB);

Console.WriteLine("Concatenated string is: {0}", str);

}

}

Output:

String A is: Hello!

String B is: World.

Concatenated string is: Hello! World.

**Explanation:**

**String.Concat Method** is used to concatenate one or more instances of [String](https://www.geeksforgeeks.org/c-string/)or the [String](https://www.geeksforgeeks.org/c-string/)representations of the values of one or more instances of Object. It always returns a concatenated string.

**Return Value:** The return type of this method is **System.String**. This method returns a string as an result of concatenation of two strings, i.e. strA, and strB.

**(b) Write about different types of type conversions available in C#.**

Type conversion is a process of converting one type into another. Using C# type conversion techniques, not only can you convert data types but you can also convert object types. Types of type conversion are discussed in the following:

* **Implicit conversions**: No special syntax is required because the conversion always succeeds and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.
* **Explicit conversions (casts)**: Explicit conversions require a [cast expression](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#cast-expression). Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.
* **User-defined conversions**: User-defined conversions are performed by special methods that you can define to enable explicit and implicit conversions between custom types that do not have a base class–derived class relationship. For more information, see [User-defined conversion operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/user-defined-conversion-operators).
* **Conversions with helper classes**: To convert between non-compatible types, such as integers and [System.DateTime](https://docs.microsoft.com/en-us/dotnet/api/system.datetime) objects, or hexadecimal strings and byte arrays, you can use the [System.BitConverter](https://docs.microsoft.com/en-us/dotnet/api/system.bitconverter) class, the [System.Convert](https://docs.microsoft.com/en-us/dotnet/api/system.convert) class, and the Parse methods of the built-in numeric types, such as [Int32.Parse](https://docs.microsoft.com/en-us/dotnet/api/system.int32.parse). For more information, see [How to convert a byte array to an int](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/how-to-convert-a-byte-array-to-an-int), [How to convert a string to a number](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/how-to-convert-a-string-to-a-number), and [How to convert between hexadecimal strings and numeric types](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/how-to-convert-between-hexadecimal-strings-and-numeric-types).

**Q3: (a) What are constants in C# discuss in detail.**

**A:** 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.

The constants are treated just like regular variables except that their values cannot be modified after their definition.

## Defining Constants

Constants are defined using the **const** keyword. Syntax for defining a constant is −

const <data\_type> <constant\_name> = value;

The following program demonstrates defining and using a constant in your program −

using System;

namespace DeclaringConstants {

class Program {

static void Main(string[] args) {

const double pi = 3.14159;

// constant declaration

double r;

Console.WriteLine("Enter Radius: ");

r = Convert.ToDouble(Console.ReadLine());

double areaCircle = pi \* r \* r;

Console.WriteLine("Radius: {0}, Area: {1}", r, areaCircle);

Console.ReadLine();

}

}

}

When the above code is compiled and executed, it produces the following result −

Enter Radius:

3

Radius: 3, Area: 28.27431

**(b) Write a program on string literal explain in detail.**

## A: String Literals

String literals or constants are enclosed in double quotes "" or with @"". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separating the parts using whitespaces.

Here are some examples of string literals. All the three forms are identical strings.

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

@"hello dear"