**Reading & Writing To The Console In C# Application**

**Input & Output In C# Program**

* Programmers often need to display the output of a C# program to users.
* The programmer can use the command line interface to display the output.
* The programmer can similarly accept inputs from a user through the command line interface.
* Such input and output operations are also known as console operations.

**Console Operations**

**Following are the features of the console operations:**

* Console operations are tasks performed on the command line interface using executable commands.
* The console operations are used in software applications because these operations are easily controlled by the operating system.
* This is because console operations are dependent on the input and output devices of the computer system.
* A console application is one that performs operations at the command prompt.
* All console applications consist of three streams, which are a series of bytes. These streams are attached to the input and output devices of the computer system and they handle the input and output operations.

**There are 3 streams:**

1. Standard in
2. Standard out
3. Standard err

**Standard in**

The standard in stream takes the input and passes it to the console application for processing.

**Standard out**

The standard out stream displays the output on the monitor.

**Standard err**

The standard err stream displays error messages on the monitor.

**Output Methods In C#**

* In C#, all console operations are handled by the Console class of the System namespace.
* A namespace is a collection of classes having similar functionalities.
* To write data on the console, you need the standard output stream, provided by the Console class.
* There are two output methods that write to the standard output stream as follows:
  + Console.Write(): Writes any type of data.
  + Console.WriteLine(): Writes any type of data and this data ends with a new line character in the standard output stream. This means any data after this line will appear on the new line.

The following syntax is used for the Console.Write() method, which allows you to display the information on the console window:

Console.Write(“<data>” + variables);

**Where,**

* data: Specifies strings or escape sequence characters enclosed in double quotes.
* variables: Specify variable names whose value should be displayed on the console.

**The following syntax is used for the Console.WriteLine() method, which allows you to display the information on a new line in the console window:**

Console.WriteLine(“<data>” + variables);

The following code shows the difference between the Console.Write() and Console.WriteLine() methods:

Console.WriteLine(“C# is a powerful programming language”);

Console.WriteLine(“C# is a powerful”);

Console.WriteLine(“programming language”);

Console.Write(“C# is a powerful”);

Console.WriteLine(“ programming language”);

**Output**

C# is a powerful programming language  
C# is a powerful  
programming language  
C# is a powerful programming language

**What Are Placeholders?**

* The WriteLine() and Write() methods accept a list of parameters to format text before displaying the output.
* The first parameter is a string containing markers in braces to indicate the position, where the values of the variables will be substituted. Each marker indicates a zero-based index based on the number of variables in the list.
* The following code uses placeholders in the Console.WriteLine() method to display the result of the multiplication operation:

int number, result;

number = 5;

result = 100 \* number;

Console.WriteLine(“Result is {0} when 100 is multiplied by {1}”, result,number);

result = 150 / number;

Console.WriteLine(“Result is {0} when 150 is divided by {1}”, +result, number);

**Output**

Result is 500 when 100 is multiplied by 5  
Result is 30 when 150 is divided by 5  
Here, {0} is replaced with the value in result and {1} is replaced with the value in number.

**Input Methods In C#**

* In C#, to read data, you need the standard input stream. This stream is provided by the input methods of the Console class. There are two input methods that enable the software to take in the input from the standard input stream.
* These methods are as follows:
  + Console.Read(): Reads a single character.
  + Console.ReadLine(): Reads a line of strings.

The following code reads the name using the ReadLine() method and displays the name on the console window:

string name;

Console.Write(“Enter your name: “);

name = Console.ReadLine();

Console.WriteLine(“You are {0}”,name);

**In Above Code,**

* The ReadLine() method reads the name as a string and the string that is given is displayed as output using placeholders.

**Output**

Enter your name: David Blake  
You are David Blake

The following code demonstrates the use of placeholders in the Console.WriteLine() method:

using System;

class Loan

{

static void Main(string[] args)

{

stringcustName;

doubleloanAmount;

float interest = 0.09F;

doubleinterestAmount = 0;

doubletotalAmount = 0;

Console.Write("Enter the name of the customer : ");

custName = Console.ReadLine();

Console.Write("Enter loan amount : ");

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

interestAmount = loanAmount \* interest;

totalAmount = loanAmount + interestAmount;

Console.WriteLine("\nCustomer Name : {0}", custName);

Console.WriteLine("Loan amount : ${0:#,###.#0} \nInterest rate : {1:0#%} \nInterest Amount : ${2:#,###.#0}",

loanAmount, interest, interestAmount );

Console.WriteLine("Total amount to be paid : ${0:#,###.#0} ", totalAmount);

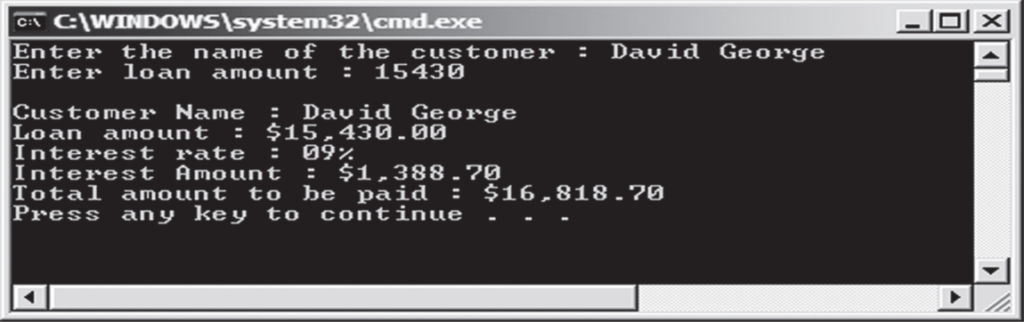
}

}

**In Above code,**

* The name and loan amount are accepted from the user using the Console.ReadLine() method. The details are displayed on the console using the Console.Writeline() method.
* The placeholders {0}, {1}, and {2} indicate the position of the first, second, and third parameters respectively.
* The 0 specified before # pads the single digit value with a 0. The #option specifies the digit position.
* The % option multiplies the value by 100 and displays the value along with the percentage sign.

**The following figure displays the example of placeholders:**



**Numeric Format Specifiers In C#**

* Format specifiers are special characters that are used to display values of variables in a particular format. For example, you can display an octal value as decimal using format specifiers.
* In C#, you can convert numeric values in different formats. For example, you can display a big number in an exponential form.
* To convert numeric values using numeric format specifiers, you should enclose the specifier in curly braces. These curly braces must be enclosed in double quotes. This is done in the output methods of the Console class.

**The following is the syntax for the numeric format specifier:**

Console.WriteLine(“{format specifier}”, <variable name>);

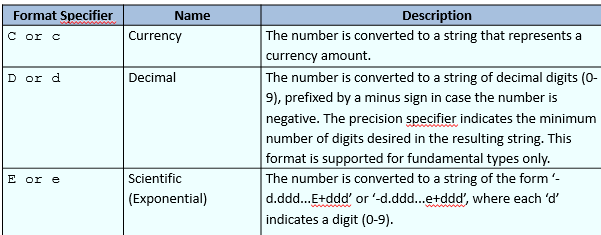
**Where,**

* formatspecifier: Is the numeric format specifier.
* variable name: Is the name of the integer variable.

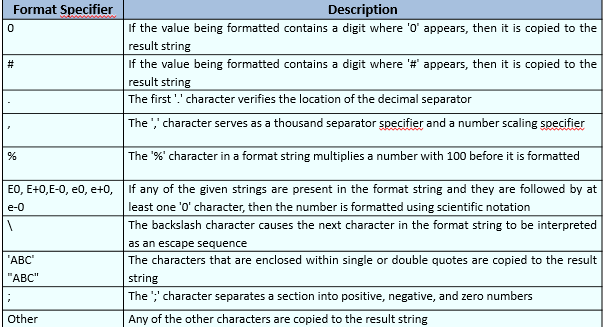
Numeric format specifiers work only with numeric data that can be suffixed with digits. The digits specify the number of zeros to be inserted, after the decimal location.

**Example**

* If you use a specifier such as C3, three zeros will be suffixed, after the decimal location of the given number. The following table lists some of the numeric format specifiers in C#:



* Custom numeric format strings contain more than one custom numeric format specifiers and define how data is formatted.
* A custom numeric format string is defined as any string that is not a standard numeric format string. The following table lists the custom numeric format specifiers and their description:



The following code demonstrates the conversion of a numeric value using C, D, and E format specifiers:

int num = 456;

Console.WriteLine(“{0:C}”, num);

Console.WriteLine(“{0:D}”, num);

Console.WriteLine(“{0:E}”, num);

**Output**

$456.00  
456  
4.560000E+002

**The following code demonstrates the use of custom numeric format specifiers:**

using System;

class Banking

{

static void Main(string[] args)

{

double loanAmount = 15590;

float interest = 0.09F;

double interestAmount = 0;

double totalAmount = 0;

interestAmount = loanAmount \* interest ;

totalAmount = loanAmount + interestAmount;

Console.WriteLine(“Loan amount : ${0:#,###.#0} “, loanAmount);

Console.WriteLine(“Interest rate : {0:0#%} “, interest);

Console.WriteLine(“Total amount to be paid :${0:#,###.#0}”,totalAmount);

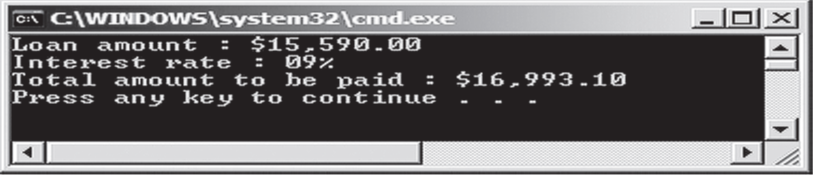
}

}

**In Above Code,**

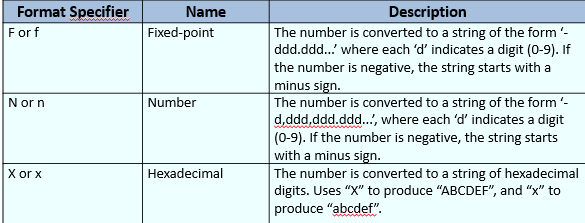
* The #, %, ., and 0 custom numeric format specifiers are used to display the loan details of the customer in the desired format.

**The following figure displays the example of custom numeric format specifiers:**



**Some More Number Format Specifiers**

There are some additional number format specifiers that are described in the following table:



The following figure demonstrates the conversion of a numeric value using F, N, and X format specifiers:

int num = 456;

Console.WriteLine(“{0:F}”, num);

Console.WriteLine(“{0:N}”, num);

Console.WriteLine(“{0:X}”, num);

**Output**

456.00  
456.00  
1C8

## Source Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace IntroductionToCsharp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter first number");

int num1 = int.Parse(Console.ReadLine());

Console.WriteLine("Enter Second number");

int num2 = int.Parse(Console.ReadLine());

int sum = num1 + num2;

Console.WriteLine("Addition result is: {0}", sum);

Console.ReadLine();

//Console.WriteLine("Enter your first Name ");

//string fname = Console.ReadLine();

//Console.WriteLine("Enter your Last Name ");

//string lname = Console.ReadLine();

//Console.WriteLine("Your Name is : " + fname + " " + lname); // concatenation syntax

//Console.WriteLine("Your Name is: {0} {1}", fname, lname); // placeholder syntax

//Console.ReadLine();

}

}

}