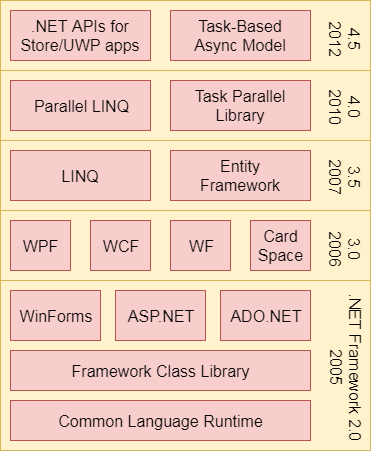
.NET Framework

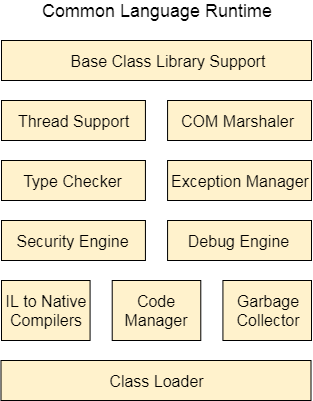
* .NET is a framework to develop software applications.
* It is designed and developed by Microsoft and the first beta version released in 2000.
* It is used to develop applications for web, Windows, phone.
* It provides a broad range of functionalities and support.
* This framework contains a large number of class libraries known as Framework Class Library (FCL).
* The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime).
* This framework provides various services like memory management, networking, security, memory management, and type-safety.
* The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.
* The .NET Framework is composed of four main components –
  + **Common Language Runtime (CLR)**
  + **Framework Class Library (FCL)**
  + **Core Languages (WinForms, ASP.NET, and ADO.NET)**
  + **Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library etc.)**



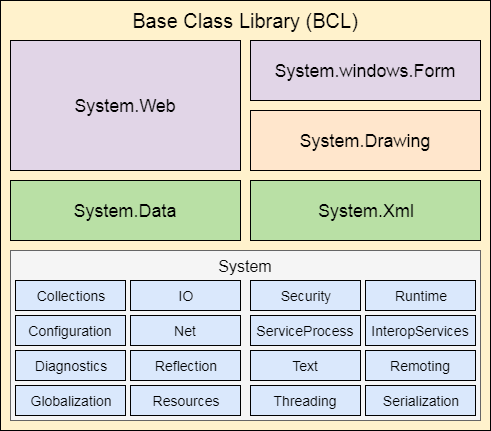
* **CLR (Common Language Runtime) –**

It is a program execution engine that loads and executes the program. It converts the program into native code. It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portability.

* **CLR components –**

****

* **FCL (Framework Class Library) –**

It is a standard library that is a collection of thousands of classes and used to build an application. The BCL (Base Class Library) is the core of the FCL and provides basic functionalities.

* **.NET Common Language Runtime (CLR) –**

.NET CLR is a run-time environment that manages and executes the code written in any .NET programming language. It converts code into native code which further can be executed by the CPU.

* **.NET CLR Functions –**

1. Handles Exceptions
2. Provides type-safety
3. Memory management
4. Provides security
5. Improved performance
6. Language independent
7. Platform independent
8. Garbage collection
9. Language features such as inheritance, interfaces, and overloading for object-oriented programming.

* **.NET CLR Versions –**

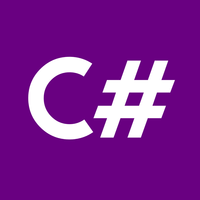
|  |  |
| --- | --- |
| **.NET version** | **CLR version** |
| 1.0 | 1.0 |
| 1.1 | 1.1 |
| 2.0 | 2.0 |
| 3.0 | 2.0 |
| 3.5 | 2.0 |
| 4 | 4 |
| 4.5 | 4 |
| 4.6 | 4 |
| 4.6 | 4 |

* **Features of CLR –**
  + **Base Class Library Support -** It is a class library that provides support of classes to the .NET application.
  + **Thread Support -** It manages the parallel execution of the multi-threaded application.
  + **COM Marshaler -** It provides communication between the COM objects and the application.
  + **Type Checker -** It checks types used in the application and verifies that they match to the standards provided by the CLR.
  + **Code Manager -** It manages code at execution run-time.
  + **Garbage Collector -** It releases the unused memory and allocates it to a new application.
  + **Exception Handler -** It handles the exception at runtime to avoid application failure.
  + **ClassLoader -** It is used to load all classes at run time.
* **.NET Framework Class Library –**

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications. It contains thousands of classes that support the following functions.

* Base and user-defined data types
* Support for exceptions handling
* Input/Output and stream operations
* Communications with the underlying system
* Access to data
* Ability to create Windows-based GUI applications
* Ability to create web-client and server applications
* Support for creating web services
* **.NET Framework Class Library Namespaces –**

|  |  |
| --- | --- |
| **Namespaces** | **Description** |
| **System** | It includes all common data types, string values, arrays and methods for data conversion. |
| **System.Data System.Data.Common System.Data.OleDb System.Data.SqlClient System.Data.SqlTypes** | These are used to access a database, perform commands on a database and retrieve database. |
| **System.IO** | These are used to access, read and write files |
| **System.Windows.Forms System.Windows.Forms.Design** | These namespaces are used to create Windows-based applications using Windows user interface components. |

****

* C# is a programming language of .Net Framework.
* **What is C# -**

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

1. Window applications
2. Web applications
3. Distributed applications
4. Web service applications
5. Database Applications etc.

**Note:**

1. **C# is approved as a standard by ECMA and ISO.**
2. **C# is designed for CLI (Common Language Infrastructure).**
3. **CLI is a specification that describes executable code and runtime environment.**
4. **C# programming language is influenced by C++, Java, Eiffel, Modula-3, Pascal etc. languages.**

* **Java vs. C# -**

**Java -**

1. Java is a high level, robust, secured and object-oriented programming language developed by Oracle.
2. Java programming language is designed to be run on a Java platform, by the help of Java Runtime Environment (JRE).
3. Java type safety is safe.
4. In java, built-in data types that are passed by value are called primitive types.
5. Arrays in Java are direct specialization of Object.
6. Java does not support conditional compilation.
7. Java doesn't support goto statement.
8. Java doesn't support structures and unions.
9. Java supports checked exception and unchecked exception.

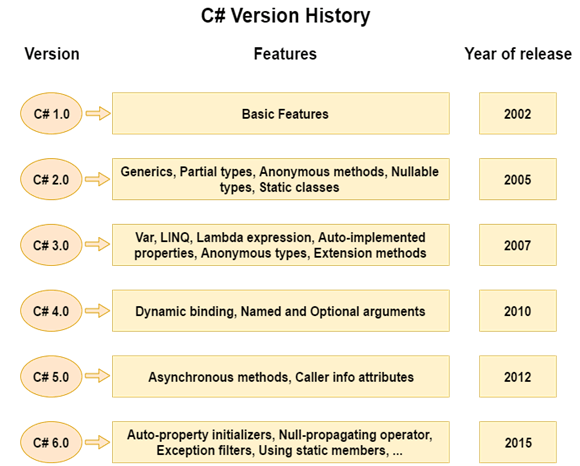
**C# -**

1. C# is an object-oriented programming language developed by Microsoft that runs on .Net Framework.
2. C# programming language is designed to be run on the Common Language Runtime (CLR).
3. C# type safety is unsafe.
4. In C#, built-in data types that are passed by value are called simple types.
5. Arrays in C# are specialization of System.
6. C# supports conditional compilation using preprocessor directives.
7. C# supports goto statement.
8. C# supports structures and unions.
9. C# supports unchecked exception.

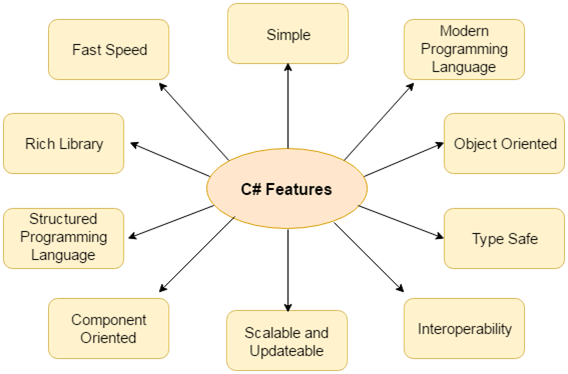
**NOTE:**

1. **C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.**
2. **Anders Hejlsberg is known as the founder of C# language.**

|  |
| --- |
| **Anders_Hejlsberg.jpg** |
| **Anders Hejlsberg** |

****

* **C# Features –**

****

* **First Example – WAP to print “Hello World” –**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class1

{

static void Main(string[] args)

{

Console.WriteLine("Hello World");

Console.ReadKey(true);

}

}

}

**Description –**

* **class:** is a keyword which is used to define class.
* **Program:** is the class name. A class is a blueprint or template from which objects are created. It can have data members and methods. Here, it has only Main method.
* **static:** is a keyword which means object is not required to access static members. So it saves memory.
* **void:** is the return type of the method. It does't return any value. In such case, return statement is not required.
* **Main:** is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.
* **string[] args:** is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.
* **System.Console.WriteLine("Hello World!"):** Here, System is the namespace. Console is the class defined in System namespace. The WriteLine() is the static method of Console class which is used to write the text on the console.
* **C# Types -**

A Type is defined as a set of data and the operations performed on them. CSharp is a **strongly typed language**. The CSharp type system contains **three Type categories**. They are - **Value Types**, **Reference Types** and **Pointer Types**. The Value Types store the data while the Reference Types store references to the actual data. Pointer Types variable use only in **unsafe mode**. **The Value Types derived from System.ValueType** and **the Reference Types derived from System.Object**.

The main difference between Value Types and Reference Types is that how these Types store the values in memory. Common Language Runtime (CLR) allocates memory in Stack and the Heap. A Value Type holds its actual value in memory allocated on the Stack and Reference Types referred to as objects, store references to the actual data.

**Note:** **In C# it is possible to convert a value of one type into a value of another type. The operation of Converting a Value Type to a Reference Type is called Boxing and the reverse operation is called Unboxing.**

* **C# Boxing and Unboxing -**

C# Type System contains three Types; they are **Value Types**, **Reference Types** and **Pointer Types**. C# allows us to convert a Value Type to a Reference Type, and back again to Value Types.

**Note: The operation of Converting a Value Type to a Reference Type is called Boxing and the reverse operation is called Unboxing.**

* **Boxing –**

The first line we created a Value Type Val and assigned a value to Val. The second line, we created an instance of Object Obj and assign the value of Val to Obj. From the above operation (**Object Obj = i**) we saw converting a value of a Value Type into a value of a corresponding Reference Type. This type of operation is called Boxing.

* **Unboxing -**

The first two lines show how to Box a Value Type. The next line (**int i = (int) Obj**) shows extracts the Value Type from the Object. That is converting a value of a Reference Type into a value of a Value Type. This operation is called Unboxing.

**Note:** **Boxing and Unboxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed, also the cast required for Unboxing is also expensive computationally.**

**Object sc = 10;** **//Boxing – Value type converts to reference type**

**int i = (int) sc;** **//Unboxing – Reference type converts to value type**

* **Type Conversion –**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**namespace BCA\_5th\_Semester**

**{**

**class Class12**

**{**

**public static void Main()**

**{**

**float sum;**

**int i;**

**sum = 0.0F;**

**for (i = 1; i <= 10; i++)**

**{**

**sum = (sum + 1) / (float)i;**

**Console.Write("i= " + i);**

**Console.WriteLine(" Sum is: " + sum);**

**}**

**Console.ReadKey();**

**}**

**}**

**}**

* **C# Access Modifiers / Specifiers –**

C# Access modifiers or specifiers are the keywords that are used to specify accessibility or scope of variables and functions in the C# application.

**C# provides five types of access specifiers –**

* Public
* Protected
* Internal
* Protected internal
* Private

|  |  |
| --- | --- |
| Access Specifier | Description |
| Public | It specifies that access is not restricted. |
| Protected | It specifies that access is limited to the containing class or in derived class. |
| Internal | It specifies that access is limited to the current assembly. |
| Protected Internal | It specifies that access is limited to the current assembly or types derived from the containing class. |
| Private | It specifies that access is limited to the containing type. |

* **Public Access Specifier –**

It makes data accessible publicly. It does not restrict data to the declared block.

**using System;**

**namespace AccessSpecifiers**

**{**

**class PublicTest**

**{**

**public string name = "Shantosh Kumar" ;**

**public void Msg(string msg)**

**{**

**Console.WriteLine( "Hello " + msg);**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**PublicTest publicTest = new PublicTest();**

**// Accessing public variable**

**Console.WriteLine( "Hello " + publicTest.name);**

**// Accessing public function**

**publicTest.Msg( "Peter Decosta" );**

**}**

**}**

**}**

* **Protected Access Specifier -**

It is accessible within the class and has limited scope. It is also accessible within sub class or child class, in case of inheritance.

**using System;**

**namespace AccessSpecifiers**

**{**

**class ProtectedTest**

**{**

**protected string name = "Shashikant" ;**

**protected void Msg(string msg)**

**{**

**Console.WriteLine( "Hello " + msg);**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**ProtectedTest protectedTest = new ProtectedTest();**

**// Accessing protected variable**

**Console.WriteLine( "Hello " + protectedTest.name);**

**// Accessing protected function**

**protectedTest.Msg( "Swami Ayyer" );**

**}**

**}**

**}**

* **Internal Access Specifier –**

The internal keyword is used to specify the internal access specifier for the variables and functions. This specifier is accessible only within files in the same assembly.

**using System;**

**namespace AccessSpecifiers**

**{**

**class InternalTest**

**{**

**internal string name = "Shantosh Kumar" ;**

**internal void Msg(string msg)**

**{**

**Console.WriteLine( "Hello " + msg);**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**InternalTest internalTest = new InternalTest();**

**// Accessing internal variable**

**Console.WriteLine( "Hello " + internalTest.name);**

**// Accessing internal function**

**internalTest.Msg( "Peter Decosta" );**

**}**

**}**

**}**

* **Protected Internal Access Specifier –**

Variable or function declared protected internal can be accessed in the assembly in which it is declared. It can also be accessed within a derived class in another assembly.

**using System;**

**namespace AccessSpecifiers**

**{**

**class InternalTest**

**{**

**protected internal string name = "Shantosh Kumar" ;**

**protected internal void Msg(string msg)**

**{**

**Console.WriteLine( "Hello " + msg);**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**InternalTest internalTest = new InternalTest();**

**// Accessing protected internal variable**

**Console.WriteLine( "Hello " + internalTest.name);**

**// Accessing protected internal function**

**internalTest.Msg( "Peter Decosta" );**

**}**

**}**

**}**

* **Private Access Specifier –**

Private Access Specifier is used to specify private accessibility to the variable or function. It is most restrictive and accessible only within the body of class in which it is declared.

**using System;**

**namespace AccessSpecifiers**

**{**

**class PrivateTest**

**{**

**private string name = "Shantosh Kumar" ;**

**private void Msg(string msg)**

**{**

**Console.WriteLine( "Hello " + msg);**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**PrivateTest privateTest = new PrivateTest();**

**// Accessing private variable**

**Console.WriteLine( "Hello " + privateTest.name);**

**// Accessing private function**

**privateTest.Msg( "Peter Decosta" );**

**}**

**}**

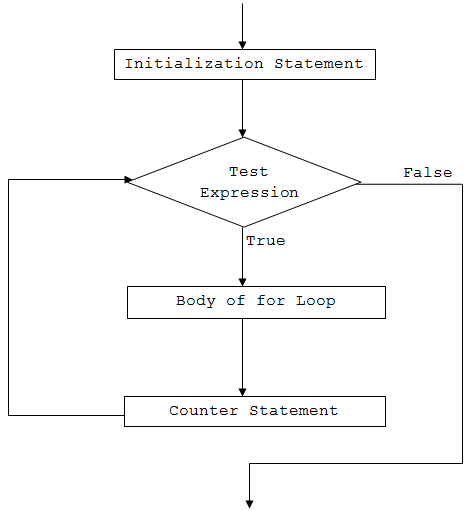
**}**

* **Loops –**

Loops are used in programming to repeatedly execute a certain block of statements until some condition is met.

* There are 3 types of loops –

1. **for loop //entry controlled loop**
2. **while loop // entry controlled loop**
3. **do-while loop //exit controlled loop**



* **For Loop –**

The for keyword indicates a loop in C#. The for loop executes a block of statements, repeatedly until the specified condition returns false.

**Syntax –**

**for (initializer; condition; iterator)**

**{**

**//code block**

**}**

The for loop contains the following three optional sections, separated by a semicolon:

* **Initializer:** The initializer section is used to initialize a variable that will be local to a for loop and cannot be accessed outside loop. It can also be zero or more assignment statements, method call, increment, or decrement expression e.g., ++i or i++, and await expression.
* **Condition:** The condition is a boolean expression that will return either true or false. If an expression evaluates to true, then it will execute the loop again; otherwise, the loop is exited.
* **Iterator:** The iterator defines the incremental or decremental of the loop variable.

**Example –**

**for(int i = 0; i < 10; i++)**

**{**

**Console.WriteLine("Value of i: {0}", i);**

**}**

* **Infinite For Loop –**

**for ( ; ; )**

**{**

**Console.Write(1);**

**}**

* **Exit the for Loop –**

We can also exit from a for loop by using the break keyword.

**for (int i = 0; i < 10; i++)**

**{**

**if( i == 5 )**

**break;**

**Console.WriteLine("Value of i: {0}", i);**

**}**

* **Nested for Loop –**

C# allows a for loop inside another for loop.

**for (int i = 0; i < 2; i++)**

**{**

**for(int j =i; j < 4; j++)**

**Console.WriteLine("Value of i: {0}, J: {1} ", i,j);**

**}**

/\* WAP to print 1 to 10 using for loop \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class13

{

public static void Main()

{

int i;

for (i = 1; i <= 10; i+=1)

{

Console.WriteLine("Value is: " + i);

}

Console.ReadKey();

}

}

}

/\* WAP to print 1 to 10 using for loop – Type – II \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class14

{

public static void Main()

{

for (int i = 1; i <= 10; i++)

{

Console.WriteLine("Value of i: {0}", i);

}

Console.ReadKey();

}

}

}

/\* Example of Infinite Loop \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class15

{

public static void Main()

{

int i;

for (i = 1; i <= 10; i--)

{

Console.WriteLine("Value is: " + i);

}

Console.ReadKey();

}

}

}

/\* WAP to print the series based on user input \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class13

{

public static void Main()

{

int i,n;

Console.WriteLine("Enter the value: ");

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

for (i = n; i >= 11; i-=1)

{

Console.WriteLine("Value is: " + i);

}

Console.ReadKey();

}

}

}

* **While Loop -**

C# provides the while loop to repeatedly execute a block of code as long as the specified condition returns false.

**Syntax –**

**while(*condition*)**

**{**

**//code block**

**}**

The while loop starts with the while keyword, and it must include a Boolean conditional expression inside brackets that returns either true or false. It executes the code block until the specified conditional expression returns false.

**Example –**

**int i = 0; // initialization**

**while (i < 10) // condition**

**{**

**Console.WriteLine("i = {0}", i);**

**i++; // increment**

**}**

* **Exit from the while Loop –**

**int i = 0;**

**while (true)**

**{**

**Console.WriteLine("i = {0}", i);**

**i++;**

**if (i > 10)**

**break;**

**}**

* **Infinite While Loop -**

**int i = 0;**

**while (i > 0)**

**{**

**Console.WriteLine("i = {0}", i);**

**i++;**

**}**

* **Nested while Loop –** C# allows while loops inside another while loop, as shown below. However, it is not recommended to use nested while loop because it makes it hard to debug and maintain.

**int i = 0, j = 1;**

**while (i < 2)**

**{**

**Console.WriteLine("i = {0}", i);**

**i++;**

**while (j < 2)**

**{**

**Console.WriteLine("j = {0}", j);**

**j++;**

**}**

**}**

/\* WAP to print 1 to 10 using for loop \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class16

{

public static void Main()

{

int i = 11;

while (i <= 10)

{

Console.WriteLine("Value is: " + i);

i++;

}

Console.ReadKey();

}

}

}

/\* WAP to print the series from 100 to 10 in reverse order \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class17

{

public static void Main()

{

int i = 100;

while (i >= 10)

{

Console.WriteLine("Value is: " + i);

i--;

}

Console.ReadKey();

}

}

}

/\* Example of Infinite Loop \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class17

{

public static void Main()

{

int i = 100;

while (i >= 10)

{

Console.WriteLine("Value is: " + i);

}

Console.ReadKey();

}

}

}

* **Do – While Loop –**
  + The C# ***do-while loop*** is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
  + The C# ***do-while loop*** is executed at least once because condition is checked after loop body.
* **Syntax -**

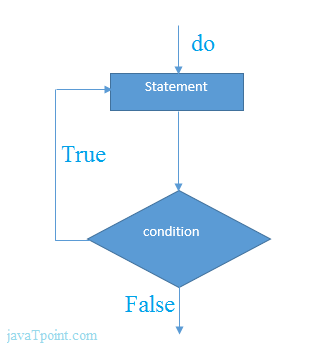
**do**

**{**

**//code to be executed**

**}**

**while(condition);**

****

* **Example –**

**using System;**

**public class DoWhileExample**

**{**

**public static void Main(string[] args)**

**{**

**int i = 1;**

**do**

**{**

**Console.WriteLine(i);**

**i++;**

**} while (i <= 10) ;**

**}**

**}**

* **Nested do-while Loop -**

In C#, if we use do-while loop inside another do-while loop, it is known as nested do-while loop. The nested do-while loop is executed fully for each outer do-while loop.

**using System;**

**public class DoWhileExample**

**{**

**public static void Main(string[] args)**

**{**

**int i=1;**

**do{**

**int j = 1;**

**do{**

**Console.WriteLine(i+" "+j);**

**j++;**

**} while (j <= 3) ;**

**i++;**

**} while (i <= 3) ;**

**}**

**}**

* **Infinite Loop Example –**

**using System;**

**public class WhileExample**

**{**

**public static void Main(string[] args)**

**{**

**do{**

**Console.WriteLine("Infinitive do-while Loop");**

**} while(true);**

**}**

**}**

**/\*WAP to show basic structure of do-while loop \*/**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class18

{

public static void Main()

{

int i = 21;

do{

Console.WriteLine("Required Value is: " + i);

}while(i<=10);

Console.ReadKey();

}

}

}

**/\* WAP to find the factorial of a number by using 3 different types of loops \*/**

* **By using For Loop -**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class19

{

public static void Main()

{

int n,i,fact = 1;

Console.WriteLine("Enter the range: ");

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

for (i = 1; i <= n; i++)

{

fact = fact \* i;

}

Console.WriteLine("Factorial is: " + fact);

Console.ReadKey();

}

}

}

* **By using While Loop -**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class20

{

public static void Main()

{

int n,i,fact = 1;

Console.WriteLine("Enter the range: ");

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

i = 1;

while (i <= n)

{

fact = fact \* i;

i++;

}

Console.WriteLine("Factorial is: " + fact);

Console.ReadKey();

}

}

}

* **By do – while Loop –**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class21

{

public static void Main()

{

int n, i, fact = 1;

Console.WriteLine("Enter the range: ");

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

i = 1;

do{

fact = fact \* i;

i++;

}while (i <= n) ;

Console.WriteLine("Factorial is: " + fact);

Console.ReadKey();

}

}

}

**/\*WAP to print the Fibonacci Sequence upto n terms\*/**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class22

{

public static void Main()

{

int n,i;

int a = 0, b = 1, sum;

Console.WriteLine("Enter the range: ");

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

Console.Write(a + " " + b);

for (i = 1; i <= (n-2); i++)

{

sum = a + b;

a = b;

b = sum;

Console.Write(" " + sum);

}

Console.ReadKey();

}

}

}

* **Switch Case Statement –**

The C# switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C#.

**switch(expression)**

**{**

**case value1: //code to be executed; break;**

**case value2: //code to be executed; break;**

**......**

**default:**

**//code to be executed if all cases are not matched; break;**

**}**

* **Example –**

**using System;**

**public class SwitchExample**

**{**

**public static void Main(string[] args)**

**{**

**Console.WriteLine("Enter a number:");**

**int num = Convert.ToInt32(Console.ReadLine());**

**switch (num)**

**{**

**case 10: Console.WriteLine("It is 10");  break;**

**case 20: Console.WriteLine("It is 20");  break;**

**case 30: Console.WriteLine("It is 30");  break;**

**default: Console.WriteLine("Not 10, 20 or 30");  break;**

**}**

**}**

**}**

* **Break Statement –**

The C# *break* is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

* ***Syntax –***

**jump-statement;**

**break;**

* **Example –**

**using System;**

**public class BreakExample**

**{**

**public static void Main(string[] args)**

**{**

**for (int i = 1; i <= 10; i++)**

**{**

**if (i == 5)**

**{**

**break;**

**}**

**Console.WriteLine(i);**

**}**

**}**

**}**

* **Continue Statement –**

The C# *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

* ***Syntax –***

**jump-statement;**

**continue;**

* **Example -**

**using System;**

**public class ContinueExample**

**{**

**public static void Main(string[] args)**

**{**

**for(int i=1;i<=10;i++)**

**{**

**if(i==5)**

**{**

**continue;**

**}**

**Console.WriteLine(i);**

**}**

**}**

**}**

* **Goto Statement –**

The C# goto statement is also known jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label. It can be used to transfer control from deeply nested loop or switch case label. Currently, it is avoided to use goto statement in C# because it makes the program complex.

**using System;**

**public class GotoExample**

**{**

**public static void Main(string[] args)**

**{**

**ineligible:**

**Console.WriteLine("You are not eligible to vote!");**

**Console.WriteLine("Enter your age:\n");**

**int age = Convert.ToInt32(Console.ReadLine());**

**if (age < 18)**

**{**

**goto ineligible;**

**}**

**else**

**{**

**Console.WriteLine("You are eligible to vote!");**

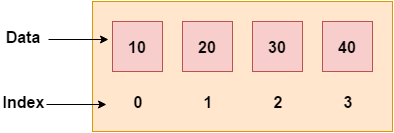
**}**

**}**

**}**

* **Arrays –**

Like other programming languages, array in C# is a group of similar types of elements that have contiguous memory location. In C#, ***array is an object of base type System.Array***. **In C#, array index starts from 0**. We can store only fixed set of elements in C# array.



**Advantages -**

* Code Optimization (less code)
* Random Access
* Easy to traverse data
* Easy to manipulate data
* Easy to sort data etc.

**Disadvantages -**

* + Fixed size

**Array Types –**

There are 3 types of arrays in C# programming –

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array (array of arrays)

**Single Dimensional Array**

**Example –**

**using System;**

**public class ArrayExample**

**{**

**public static void Main(string[] args)**

**{**

**int[ ] arr = new int[5]; //creating array**

**arr[0] = 10; //initializing array**

**arr[2] = 20;**

**arr[4] = 30;**

**//traversing array**

**for (int i = 0; i < arr.Length; i++)**

**{**

**Console.WriteLine(arr[i]);**

**}**

**}**

**}**

**Example –**

**using System;**

**public class ArrayExample**

**{**

**public static void Main(string[] args)**

**{**

**int[] arr = { 10, 20, 30, 40, 50 }; //Declaration and Initialization of array**

**//traversing array**

**for (int i = 0; i < arr.Length; i++)**

**{**

**Console.WriteLine(arr[i]);**

**}**

**}**

**}**

**Traversal using foreach loop –**

**using System;**

**public class ArrayExample**

**{**

**public static void Main(string[] args)**

**{**

**int[] arr = { 10, 20, 30, 40, 50 };//creating and initializing array**

**//traversing array**

**foreach (int i in arr)**

**{**

**Console.WriteLine(i);**

**}**

**}**

**}**

**Passing Array to Function –**

In C#, to reuse the array logic, we can create function. To pass array to function in C#, we need to provide only array name.

**Example –**

**using System;**

**public class ArrayExample**

**{**

**static void printArray(int[] arr)**

**{**

**Console.WriteLine("Printing array elements:");**

**for (int i = 0; i < arr.Length; i++)**

**{**

**Console.WriteLine(arr[i]);**

**}**

**}**

**public static void Main(string[] args)**

**{**

**int[] arr1 = { 25, 10, 20, 15, 40, 50 };**

**int[] arr2 = { 12, 23, 44, 11, 54 };**

**printArray(arr1);//passing array to function**

**printArray(arr2);**

**}**

**}**

**Example - I**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class25

{

public static void Main()

{

int i;

//creating array

int[] arr = new int[5];

//initializing array

arr[3] = 10;

arr[0] = 500;

for (i = 0; i < arr.Length; i++)

{

if(arr[i] != 0)

Console.WriteLine("Value of array: " + arr[i]);

}

Console.ReadKey();

}

}

}

**Example - II**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class26

{

public static void Main(string[] args)

{

int[] arr = { 10, 20, 30, 40, 50 };

//traversing array

for (int i = arr.Length; i > 3; i--)

{

Console.Write(arr[i-1] + " ");

}

Console.ReadKey();

}

}

}

**Example - III**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class27

{

public static void Main()

{

int i;

int[] arr = new int[5];

// user input from console

Console.WriteLine("Enter the elements: ");

for (i = 0; i < arr.Length; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

//print the values

Console.WriteLine("Value of the elements are -");

for (i = 0; i < arr.Length; i++)

{

Console.Write(arr[i] + " ");

}

Console.ReadKey();

}

}

}

**Example - IV**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class28

{

public static void Main()

{

int i,sum = 0;

int[] arr = new int[5];

// user input from console

Console.WriteLine("Enter the elements: ");

for (i = 0; i < arr.Length; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

//Sum of array elements

for (i = 0; i < arr.Length; i++)

{

sum = sum + arr[i];

}

Console.WriteLine("Sum of array elemets is: " + sum);

foreach (int j in arr)

{

Console.WriteLine(j);

}

Console.ReadKey();

}

}

}

/\* WAP to print the array in reverse direction \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class29

{

public static void Main()

{

int i;

int[] arr = new int[5];

// user input from console

Console.WriteLine("Enter the elements: ");

for (i = 0; i < arr.Length; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

// print the array in reverse order

Console.WriteLine("-- Reverse Order --");

for (i = (arr.Length-1); i >= 0; i--)

{

Console.Write(arr[i] + " ");

}

Console.ReadKey(true);

}

}

}

/\* WAP to pass an array as a parameter of a function \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class30

{

static void printArray(int[] arr)

{

Console.WriteLine("Printing array elements:");

for (int i = 0; i < arr.Length; i++)

{

Console.WriteLine(arr[i]);

}

}

public static void Main()

{

int[] arr1 = { 25, 10, 20, 15, 40, 50 };

printArray(arr1);

Console.ReadKey(true);

}

}

}

/\* WAP to print the prime numbers from an array \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class31

{

int[] arr = new int[5];

void input()

{

int i;

Console.WriteLine("Enter the elements: ");

for (i = 0; i < arr.Length; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

}

void chkPrime(int[] p)

{

int i,j,temp,c=0;

for (i = 0; i < p.Length; i++)

{

temp = p[i];

c = 0;

for (j = 1; j <= temp; j++)

{

if (temp % j == 0)

{

c++;

}

}

if (c == 2)

{

Console.Write(temp + " ");

}

}

}

public static void Main()

{

Class31 obj = new Class31();

obj.input();

Console.WriteLine("Prime Numbers are: ");

obj.chkPrime(obj.arr);

Console.ReadKey();

}

}

}

/\* WAP to print -- Linear Search \*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace BCA\_5th\_Semester

{

class Class32

{

int[] arr = new int[5];

int val;

void input()

{

int i;

Console.WriteLine("Enter the elements: ");

for (i = 0; i < arr.Length; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

Console.WriteLine("Enter the checking value: ");

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

}

void srch(int[] p)

{

int i,temp,flag = 0;

for (i = 0; i < p.Length; i++)

{

temp = p[i];

if (temp == val)

{

flag = 1;

break;

}

}

if (flag == 1)

{

Console.WriteLine("Search is successful");

}

else

{

Console.WriteLine("Search is not successful");

}

}

public static void Main()

{

Class32 obj = new Class32();

obj.input();

obj.srch(obj.arr);

Console.ReadKey();

}

}

}

* **Multidimensional Arrays –**

The multidimensional array is also known as rectangular arrays in C#. It can be two dimensional or three dimensional. The data is stored in tabular form (row \* column) which is also known as matrix.

To create multidimensional array, we need to use comma inside the square brackets.

**Syntax - int[,] arr=new int[3,3]; //declaration of 2D array**

**Example –**

**using System;**

**public class MultiArrayExample**

**{**

**public static void Main(string[] args)**

**{**

**int[,] arr=new int[3,3];//declaration of 2D array**

**arr[0,1]=10;//initialization**

**arr[1,2]=20;**

**arr[2,0]=30;**

**//traversal**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**Console.Write(arr[i,j]+" ");**

**}**

**Console.WriteLine();//new line at each row**

**}**

**}**

**}**

* **Multidimensional Array Example: Declaration and initialization at same time –**

There are 3 ways to initialize multidimensional array in C# while declaration.

1. **int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };**
2. **int[,] arr = new int[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };**
3. **int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };**

**Example –**

**using System;**

**public class MultiArrayExample**

**{**

**public static void Main(string[] args)**

**{**

**int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };**

**//traversal**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**Console.Write(arr[i,j]+" ");**

**}**

**Console.WriteLine();//new line at each row**

**}**

**}**

**}**