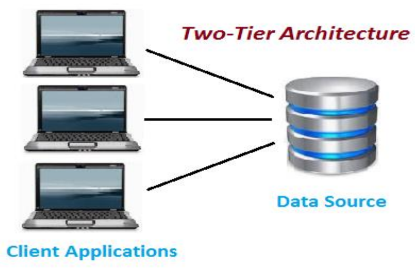
What is Client -server architecture?

Client/server architecture is a computing model in which the **server** hosts, delivers and manages most of the resources and services to be consumed by the **client**.

There are two types of client-server architecture.
1. Two tier architecture
2. Three tier architecture

Two tier client server architecture: The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.



Two-Tier Architecture

The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.

The Two-tier architecture is divided into two parts:

1) Client Application (Client Tier)
2) Database (Data Tier)

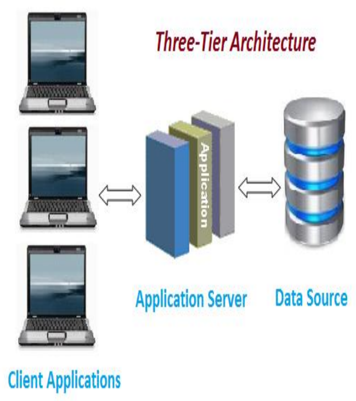
On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.

Advantages:
Easy to maintain and modification is bit easy
Communication is faster

Disadvantages:
Application performance will be degrade upon increasing the users.
Cost-ineffective

Three-Tier Architecture:
Three-tier architecture typically comprise a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows:

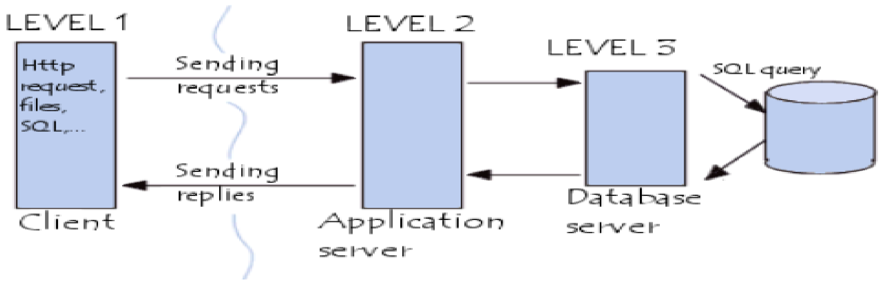
1) Client layer
2) Business layer
3) Data layer



1) Client layer:
It is also called as Presentation layer which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button, etc..

2) Business layer:
In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:
In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.



[Three Tier Architecture]

Advantages
High performance
Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
High degree of flexibility in deployment platform and configuration
Better Re-use
Improve Data Integrity
Improved Security – Client is not direct access to database.
Easy to maintain and modification is bit easy, won’t affect other modules
application performance is good.

Disadvantages
Increase Complexity/Effort

Client Vs Server

Client	Server
• A client machine is a small computer with a basic hardware configuration.	• A server machine is a high-end computer with an advanced hardware configuration.
• A client is a simple and less powerful machine .	• A server is a powerful expensive machine.
• A client is used for simple tasks .	• A server is used for storing huge data files and applications.
• A Client delivers low performance compared to a server.	• A server delivers high performance compared to a client machine.
• A client supports a single user log-in at a time.	• A server supports simultaneous, multiple user log-ins.
• A client is a program, or machine, that sends requests to servers.	• A server is a program, or machine, that waits for incoming requests.

ASP.Net Essentials:

Comparison between .NET Framework 2.0 and .NET Framework 4.0

.NET Framework 2.0	.NET Framework 4.0
• It was released in 2005	• It was released in 2010
• Development Application name is 'Visual Studio 2005'	• Development Application name is 'Visual Studio 2010'
• Main element is FCL (Framework Class Library)	• Main element is parallel LINQ (Language INtegrated Query) and Dynamic Library support
• In reference of ASP.NET application we can not change source language at runtime	• In reference of ASP.NET application we can change source language at runtime
• Parallel and cloud computing process not support.	• Parallel and cloud computing process support.
• At client level WPF (Windows Presentation Foundation) and WWF (Windows Workflow Foundation) not stimulate with CLR.	• At client level WPF (Windows Presentation Foundation) and WWF (Windows Workflow Foundation) stimulate with CLR.
• At Communication level WCF (Windows Communication Foundation) not support so the emerging technology like MVC (Model View Control) not support.	• At Communication level WCF (Windows Communication Foundation) support so the emerging technology like MVC (Model View Control) full support.
• In process Side-By-Side not support so multiple version of .NET can't activate at same framework.	• In process Side-By-Side full support so multiple version of .NET can easily activate at same framework.
• AJAX control doesn't have separate library.	• AJAX control has separate library named MAL (Microsoft AJAX Library)
• In short Framework 2.0 is Generics, Partial classes, anonymous methods, nullable types.	• In short Framework 4.0 is MVC based, Parallel classes, Dynamic Library methods, rich data types support.

Developing a Web Application:

What is ASP.NET?

ASP.NET is a web application framework developed by Microsoft to build dynamic data driven web application and web services
ASP.NET is a subset of .NET framework.
A framework is a collection of classes.
.NET framework is used to build variety of applications.
1. Console Application
2. Windows Application
3. Web application
4. Web services… etc.

ASP.NET is introduced in year 2002 with .NET framework 1.0.
Prior to ASP.NET, classic ASP (Active Server Pages) is used to build applications.
ASP.NET is the successor to classic ASP.

What is Web Applications?

A web application is an application that is accessed by users using web browser. Exam- If we want to search something from the internet. Then someone open a browser and type google.com in it. So google.com is an example of web application.

There are different browsers available.

1. Microsoft Internet Explorer
2. Google Chrome
3. Mozilla Firefox
4. Apple Safari
5. Netscape Navigator

There are alternate technologies available, which we can use to build web applications.
Exa-

1. PHP
2. Java
3. CGI
4. Ruby on rail
5. Perl and many more.

What are the advantages of web applications over the desktop applications?

There are many advantages of web applications over desktop applications, which are following:
1. Web applications just need to be installed only on the web server, where as desktop applications need to be installed on every computer, where you want to access them.
2. Maintenance, support and patches are easier to provide.
3. Only a browser is required on the client machine to access a web application.
4. Accessible from anywhere, provided there is internet.
5. Cross Platform.

What is a Page Directive?

Page Directives are commands. These commands are used by the compiler when the page is compiled.

How to use the directives in an ASP.NET page

It is not difficult to add a directive to an ASP.NET page. It is simple to add directives to an ASP.NET page. You can write directives in the following format:

```
<%@[Directive][Attributes]%>
```

See the directive format, it starts with "<%@" and ends with "%>". The best way is to put the directive at the top of your page. But you can put a directive anywhere in a page. One more thing, you can put more than one attribute in a single directive.

@Page

When you want to specify the attributes for an ASP.NET page then you need to use @Page Directive. As you know, an ASP.NET page is a very important part of ASP.NET, so this directive is commonly used in ASP.NET.

Example:

```
<%@Page Language="C#" AutoEventWIreup="false" CodeFile="Default.aspx.cs" Inherits="_Default"%>
```

Basics of C# Variables:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.

The basic value types provided in C# can be categorized as −

Type	Example
Integral types	sbyte, byte, short, ushort, int, uint, long, ulong, and char
Floating point types	float and double
Decimal types	decimal
Boolean types	true or false values, as assigned
Nullable types	Nullable data types

Defining Variables

Syntax for variable definition in C# is −

```
<data_type><variable_list>;
```

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here –

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

You can initialize a variable at the time of definition as –

```
int i = 100;
```

Initializing Variables

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is –

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as –

```
<data_type><variable_name> = value;
```

Some examples are –

```
int d =3, f =5;/* initializing d and f. */

byte z =22;/* initializes z. */

double pi =3.14159;/* declares an approximation of pi. */

char x ='x';/* the variable x has the value 'x'. */
```

Accepting Values from User

The **Console** class in the **System** namespace provides a function **ReadLine()**for accepting input from the user and store it into a variable.
For example,

```
int num;

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

                                    Or

Num = int.Parse(Console.ReadLine());
```

The function **Convert.ToInt32()** converts the data entered by the user to int data type, because **Console.ReadLine()** accepts the data in string format.

Operators in C#:

Operators are the foundation of any programming language. C#, operators Can be categorized **based upon** their different **functionality** :

- **Arithmetic Operators**
- **Relational Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Assignment Operators**
- **Conditional Operator**

In C#, Operators can also categorized **based upon Number of Operands :**

- **Unary Operator :** Operator that takes **one** operand to perform the operation.
- **Binary Operator :** Operator that takes **two** operands to perform the operation.
- **Ternary Operator :** Operator that takes **three** operands to perform the operation.

Arithmetic Operators

Following table shows all the arithmetic operators supported by C#. Assume variable **A** holds 10 and variable **B** holds 20 then −

Examples

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Relational / Comparison Operator

Relational operators are used to check the relationship between two operands. If the relationship is true the result will be `true`, otherwise it will result in `false`.

Relational operators are used in decision making and loops.

C# Relational Operators

Operator	Operator Name	Example
==	Equal to	6 == 4 evaluates to false
>	Greater than	3 > -1 evaluates to true
<	Less than	5 < 3 evaluates to false
>=	Greater than or equal to	4 >= 4 evaluates to true
<=	Less than or equal to	5 <= 3 evaluates to false
!=	Not equal to	10 != 2 evaluates to true

Logical Operators:

Logical operators are used to perform logical operation such as `and`, `or`. Logical operators operates on boolean expressions (`true` and `false`) and returns boolean values. Logical operators are used in decision making and loops.

Here is how the result is evaluated for logical `AND` and `OR` operators.

C# Logical operators

Operand 1	Operand 2	OR (||)	AND (&&)
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

Bitwise Operators

Bitwise operator works on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; then in the binary format they are as follows −

A = 0011 1100

B = 0000 1101

-------------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C# are listed in the following table. Assume variable A holds 60 and variable B holds 13, then −

Examples:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, which is 1100 0011 in 2's complement due to a signed binary number.

<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

Assignment Operators :

There are following assignment operators supported by C# −

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Conditional Operator

It is ternary operator which is a shorthand version of if-else statement. It has three operands and hence the name ternary. It will return one of two values depending on the value of a Boolean expression.

Syntax :

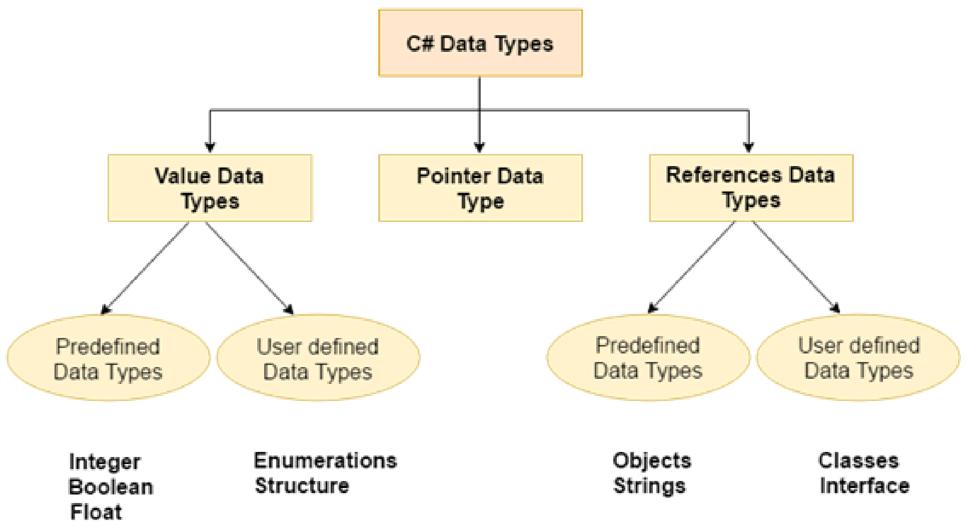
```
condition ? first_expression : second_expression;
```

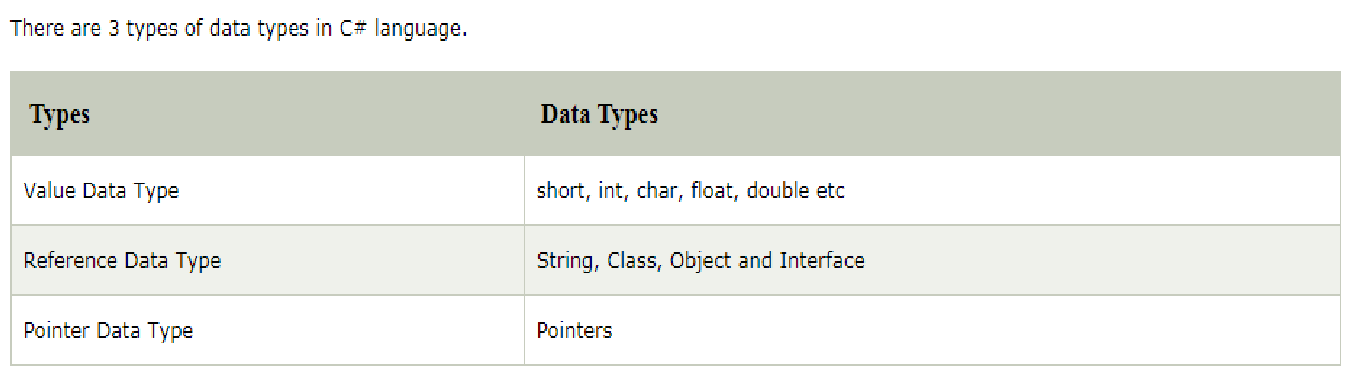
Explanation :
condition: It must be evaluate to true or false.
If the condition is true
first_expression is evaluated and becomes the result.
If the condition is false,
second_expression is evaluated and becomes the result.

--

C# Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.





There are 3 types of data types in C# language.

Types	Data Types
Value Data Type	short, int, char, float, double etc
Reference Data Type	String, Class, Object and Interface
Pointer Data Type	Pointers

There are 2 types of value data type in C# language.

1) Predefined Data Types - such as Integer, Boolean, Float, etc.

2) User defined Data Types - such as Structure, Enumerations, etc.

The memory size of data types may change according to 32 or 64 bit operating system.

Let's see the value data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to -2,147,483,647
signed int	4 byte	-2,147,483,648 to -2,147,483,647
unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

unsigned long	8 byte	0 - 18,446,744,073,709,551,615
float	4 byte	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$, 7-digit precision
double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$, 15-digit precision
decimal	16 byte	at least $-7.9 * 10^{?28}$ - $7.9 * 10^{28}$, with at least 28-digit precision

Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

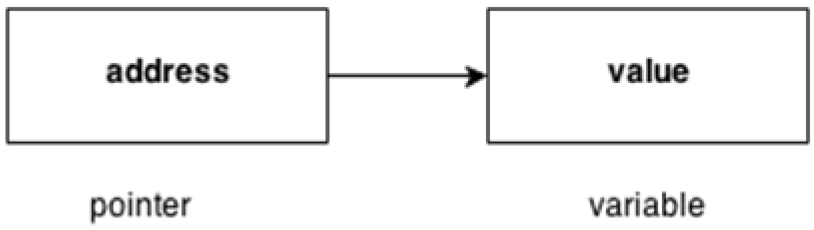
There are 2 types of reference data type in C# language.

1) Predefined Types - such as Objects, String.

2) User defined Types - such as Classes, Interface.

Pointer Data Type

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.



Control Structure:
C# offers three types of control statement:
1. Selection Statement : This consosts of if, else, switch and case branching
2. Iteration Statement: This consists of do, for, foreach, and while looping
3. Jump Statement: This consists of break, continue and return and goto statement

If Statement:
The **if** statement determines which statement to execute based on a specified condition. It has three forms

1. Single selection
2. If..then else and
3. Multicase

Single selection:
Syntax:

```
 if(boolean expression)
{
      //Statement;
 }
```

Example:

```
Static void Main()
{
         Console.WriteLine("Please enter the number:");
         int no = int.Parse (Console.ReadLine());

         if(no>0)
         {
            Console.WriteLine("Number is positive");
         }
         Console.ReadLine();
}
```

OutPut:

```
Please enter the number: 4
Number is positive
```

If..then else:
The if statement in C# may have an optional else statement. The block of code inside the else statement will be executed if the expression is evaluated to `false`.
Syntax:

```
If(boolean expression)
{
          //Statement;
}
else
{
          //Statement;
}
```

Example:

```
staticvoid Main()
{
        Console.WriteLine("Please enter the number:");
        int no = int.Parse (Console.ReadLine());

        if(no % 2 == 0)
         {
                Console.WriteLine("{0} is Even number", no);
        }
        else
         {
                Console.WriteLine("{0} is Odd Number", no);
        }

                Console.ReadLine();
 }
```

Output:

```
Please enter the number: 15
15 is Odd Number
```

Nestead (Multiple) if..else statement:

In c#, **Nested if-else** statements or conditions are useful to include one if...else statement within another if...else statement to test one condition followed by another condition.

Syntax:

```
If(boolean expression)
{
        //Statement;
}
else if(boolean expression)
{
        //Statement;
}
Else
{
        //Statement;
}
```

Example: Find maximum from three entered number.

```
classProgram
{
        staticvoid Main(string[] args)
        {
                int a, b, c;
                Console.WriteLine("enter first number:");
                 a = Convert.ToInt32(Console.ReadLine());

                Console.WriteLine("enter second number:");
                b = Convert.ToInt32(Console.ReadLine());

                Console.WriteLine("enter third number:");
                 c = Convert.ToInt32(Console.ReadLine());

        if (a > b && a > c)
        {
                Console.WriteLine("The bigger number is {0}", a);
                Console.ReadLine();
         }
        elseif (b > a && b > c)
        {
                Console.WriteLine("The bigger number is {0}", b);
                Console.ReadLine();
        }
        else
        {
                Console.WriteLine("The bigger number is {0}", c);
                Console.ReadLine();
        }
        }
 }
```

Output:

```
Enter first number: 45
Enter second number: 60
Enter third number: 5
The bigger number is 60
```

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 (variable initialization; condition; steps)
{
        //execute this code block as long as condition is satisfied
}
```

As per the syntax above, the for loop contains three parts: initialization, conditional expression and steps, which are separated by a semicolon.

1. variable initialization: Declare & initialize a variable here which will be used in conditional expression and steps part.
2. condition: The condition is a boolean expression which will return either true or false.
3. steps: The steps defines the incremental or decremental part

Example:

```
staticvoid Main(string[] args)
{
        int i,n ;
        Console.WriteLine("Enter number:");
         n = int.Parse(Console.ReadLine());

        // Print 1 to 5
        for (i = 1; i <= n; i++)
        {
                Console.WriteLine(i + " ");
        }
          Console.ReadLine();
}
```

Output:

```
1 2 3 4 5
```

Switch loop:

C# includes another decision making statement called switch. The switch statement executes the code block depending upon the resulted value of an expression.

Syntax:

```
switch(expression)
{
   case <value1>:
                  // code block
                   break;
   case <value2>:
                    // code block
                  break;
   case <valueN>:
                   // code block
                  break;
   default :
                   // code block
                  break;
}
```

As per the syntax above, switch statement contains an expression into brackets. It also includes multiple case labels, where each case represents a particular literal value. The switch cases are seperated by a break keyword which stops the execution of a particular case. Also, the switch can include a default case to execute if no case value satisfies the expression.

Example:

```
int x = 10;
switch (x)
{
case 5:
                  Console.WriteLine("Value of x is 5");
                  break;
case 10:
                  Console.WriteLine("Value of x is 10");
                  break;
case 15:
                  Console.WriteLine("Value of x is 15");
                  break;
default:
                  Console.WriteLine("Unknown value");
                  break;
}
```

Output:

```
Value of x is 10
```

While loop:
C# includes the while loop to execute a block of code repeatedly.

Syntax:

```
While(boolean expression)
{
        //execute code as long as condition returns true

}
```

As per the while loop syntax, the while loop includes a boolean expression as a condition which will return true or false. It executes the code block, as long as the specified conditional expression returns true. Here, the initialization should be done before the loop starts and increment or decrement steps should be inside the loop.

Example:

```
int i = 0;
while (i < 10)
{
Console.WriteLine("Value of i: {0}", i);

  i++;
}
```

It will print:

```
Value of i: 0
Value of i: 1
Value of i: 2
Value of i: 3
Value of i: 4
Value of i: 5
Value of i: 6
Value of i: 7
Value of i: 8
Value of i: 9
```

Do..While loop:
The do-while loop is the same as a 'while' loop except that the block of code will be executed at least once, because it first executes the block of code and then it checks the condition.

Syntax:

```
do
{
   //execute code block

} while(boolean expression);
```

As per the syntax above, do-while loop starts with the 'do' keyword followed by a code block and boolean expression with 'while'.

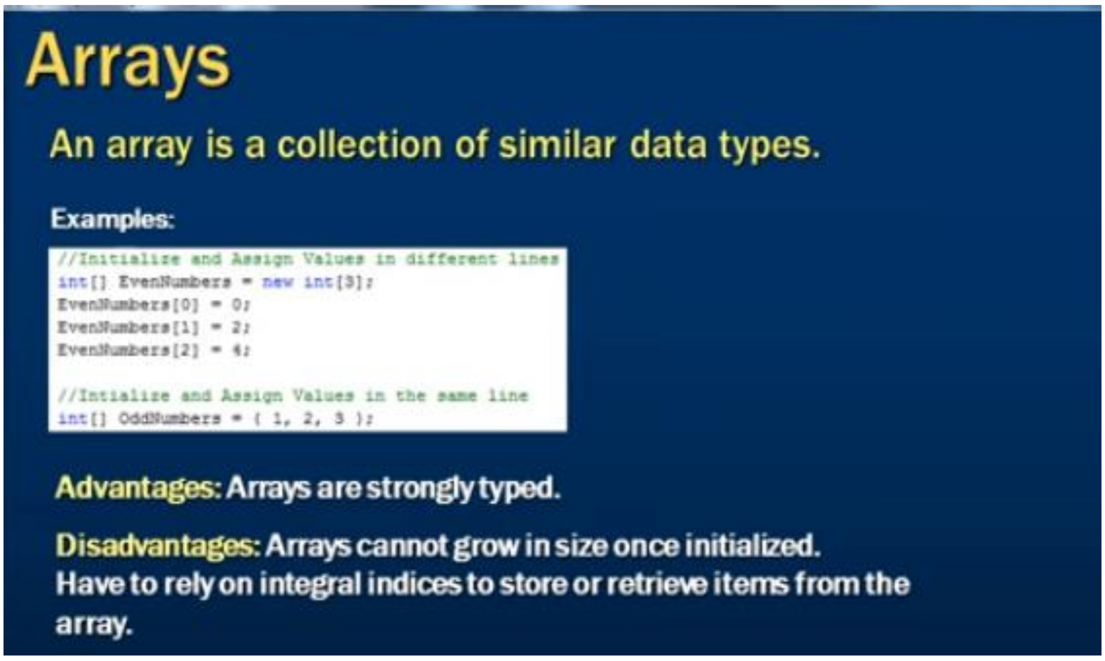
Example:

```
int i = 0;
do
{
   Console.WriteLine("Value of i: {0}", i);

   i++;

} while (i < 10);
```

It will print:

```
Value of i: 0
Value of i: 1
Value of i: 2
Value of i: 3
Value of i: 4
Value of i: 5
Value of i: 6
Value of i: 7
Value of i: 8
Value of i: 9
```

Array in C#:



Arrays

An array is a collection of similar data types.

Examples:

```
//Initialize and Assign Values in different lines
int[] EvenNumbers = new int[3];
EvenNumbers[0] = 0;
EvenNumbers[1] = 2;
EvenNumbers[2] = 4;

//Intialize and Assign Values in the same line
int[] OddNumbers = { 1, 2, 3 };
```

Advantages: Arrays are strongly typed.

Disadvantages: Arrays cannot grow in size once initialized.
Have to rely on integral indices to store or retrieve items from the array.

An array is a reference type so the **new** keyword used to create an instance of the array. We can assign initialize individual array elements, with the help of the index.

Syntax :

```
type [ ] < Name_Array > = new < datatype > [size];
```

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and Name_Array is the name of array variable. And **new** will allocate memory to an array according to its size.

Examples : To Show Different ways for the Array Declaration and Initialization

Example 1 :

```
// defining array with size 5.

// But not assigns values

int[] intArray1 = new int[5];
```

Above statement declares & initializes int type array that can store five int values. The array size is specified in square brackets([]).

Example 2 :

```
// defining array with size 5 and assigning

// values at the same time

int[] intArray2 = new int[5]{1, 2, 3, 4, 5};
```

In the above statement is same as, but it assigns values to each index in {}.

Example 3 :

```
// defining array with 5 elements which

// indicates the size of an array

int[] intArray3 = {1, 2, 3, 4, 5};
```

In above statement, the value of the array is directly initialized without taking its size. So, array size will automatically be the number of values which is directly taken.

Initialization of an Array after Declaration

Arrays can be initialized after the declaration. It is not necessary to declare and initialize at the same time using the **new** keyword. However, Initializing an Array after the declaration, it must be initialized with the new keyword. It can’t be initialized by only assigning values.

Example :

```
// Declaration of the array
string[] str1, str2;

// Intialization of array
str1 = new string[5]{ “Element 1”, “Element 2”, “Element 3”, “Element 4”, “Element 5” };

str2 = new string[]{ “Element 1”, “Element 2”, “Element 3”, “Element 4”, “Element 5” };
```

Note : Initialization without giving size is not valid in C#. It will give compile time error.

Example : Wrong Declaration for initializing an array

```
// compile-time error: must give size of an array
int[] intArray = new int[];

// error : wrong initialization of an array
string[] str1;
str1 = {“Element 1”, “Element 2”, “Element 3”, “Element 4” };
```

Foreach loop:
The foreach statement in C# iterates through a collection of items such as an array or list, The foreach bod must be enclosed in {} braces unless it consists of a single statement.

The code in listing 1 creates an array of odd numbers and uses foreach loop to loop through the array items and read them.

Example:

```
    static void Main(string[] args)
  {

    int[] Numbers = new int[3];

    Numbers[0] = 101;
    Numbers[1] = 102;
    Numbers[2] = 103;


    foreach (int k in Numbers)
    {
      Console.WriteLine(k);
    }
    Console.Read();
  }
```

Output:

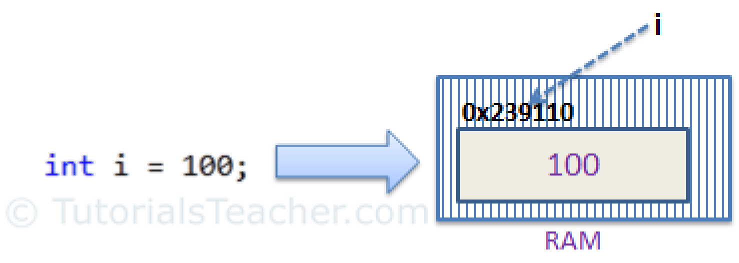
```
101
102
103
```

Types-Value types and reference types:

Value types:
A data type is a value type if it holds a data value within its own memory space. It means variables of these data types directly contain their values.

For example, consider integer variable int i = 100;

The system stores 100 in the memory space allocated for the variable 'i'. The following image illustrates how 100 is stored at some hypothetical location in the memory (0x239110) for 'i':



Memory allocation for Value Type

The following data types are all of value type:

- bool
- byte
- char
- decimal
- double
- enum
- float
- int
- long
- sbyte
- short
- struct
- uint
- ulong
- ushort

Passing by Value:
When you pass a value type variable from one method to another method, the system creates a separate copy of a variable in another method, so that if value got changed in the one method won't affect on the variable in another method.

Example:
```
static void Main(string[] args)
 {

     int x = 24;
     Console.WriteLine("X= " + x);

     f2(x);
     Console.WriteLine("X= " + x);
     Console.ReadLine();
 }
 static void f2(int k)
 {
     k = k + 10;
 }
```

Output:
X= 24
X= 24

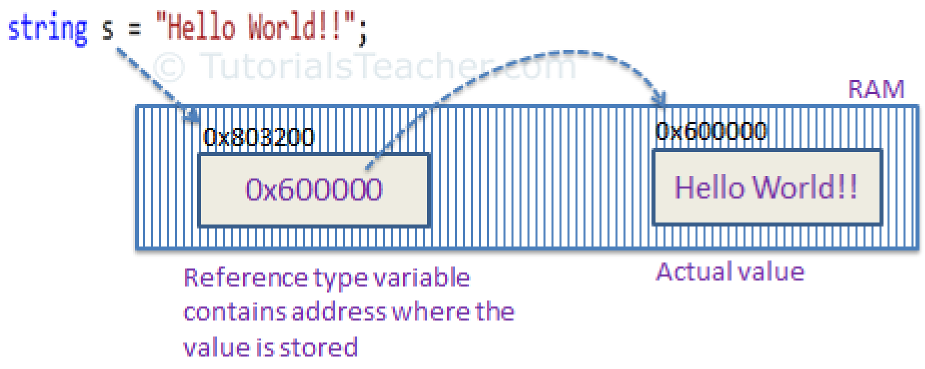
Reference Type

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

For example, consider following string variable:

```
string s = "Hello World!!";
```

The following image shows how the system allocates the memory for the above string variable.


string s = "Hello World!!";
© TutorialsTeacher.com
RAM
0x803200
0x600000
0x600000
Hello World!!
Reference type variable contains address where the value is stored
Actual value

Memory allocation for Reference type

As you can see in the above image, the system selects a random location in memory(0x803200) for the variable 's'. The value of a variable s is 0x600000 which is the memory address of the actual data value. Thus, reference type stores the address of the location where the actual value is stored instead of value itself.

The following data types are of reference type:

- String
- All arrays, even if their elements are value types
- Class
- Delegates

Pass by Reference:
When you pass a reference type variable from one method to another, it doesn't create a new copy; instead, it passes the address of the variable. If we now change the value of the variable in a method, it will also be reflected in the calling method.

Example:
```
class A
 {
     public int x;
 }
 static void Main(string[] args)
 {

     A v1 = new A();
     v1.x = 34;

     Console.WriteLine("Before= " + v1.x);

     f2(v1);
     Console.WriteLine("After= " + v1.x);
     Console.ReadLine();
 }
 static void f2(A v2)
 {
     v2.x = v2.x + 10;
 }
```

Output:
Before:=34
After= 44

Namespace:
A namespace declaration, using System indicates that you are using a System namespace.
A namespace is used to organize your code and is collection of classes, interfaces, stucts, enums and delegates.
Namespaces are used to provide " a named space" in which your application resides.
In other words, namespace is logical group of types. It is also a container.
There is two way to call namespace members -

1. By implementing fully qualified name inline :
   If we don't want to declare namespace at the top then we can declare it in our code.

   Example:
   ```
   System .Console .WriteLine ("Welcome to C#");
   ```

2. By implementing the using directive:

   Example:
   ```
   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;
   ```