C# Fundamentals

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

Boxing

```
1:    int Val = 1;
2:    Object Obj = Val; //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 . These types of operation is called Boxing.

UnBoxing

```
1:    int Val = 1;
2:    Object Obj = Val; //Boxing
3:    int i = (int)Obj; //Unboxing
```

The first two line shows 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.

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.

```
private void button1_Click(object sender, EventArgs e)
{
   int Val = 1;
   Object Obj = Val;                //Boxing
   int i = (int)Obj;                //Unboxing
   Console.WriteLine("The value is   : " + i);
}
```

C# DataTypes

Data Types in a programming language describes that what type of data a variable can hold . CSharp is a strongly typed language, therefore every variable and object must have a declared type. The CSharp type system contains three Type categories. They are Value Types , Reference Types and Pointer Types . In CSharp 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 .

When we declare a variable, we have to tell the compiler about what type of the data the variable can hold or which data type the variable belongs to.

Syntax : DataType VariableName
DataType : The type of data that the variable can hold
VariableName : the variable we declare for hold the values.

Example:

```
int count;
```
int : is the data type
count : is the variable name

The above example shows , declare a variable 'count' for holding an integer values.

The following are the commonly using datatypes in C# .

bool

The bool keyword is an alias of System.Boolean. It is used to declare variables to store the Boolean values, true and false. In C# , there is no conversion between the bool type and other types.

C# Runtime type : System.Boolean
CSharp declaration : bool flag;
CSharp Initialization : flag = true;
CSharp default initialization value : false

int

int variables are stored signed 32 bit integer values in the range of -2,147,483,648 to +2,147,483,647

C# Runtime type : System.Int32
CSharp declaration : int count;
CSharp Initialization : count = 100;
CSharp default initialization value : 0

decimal

The decimal keyword denotes a 128-bit data type.The approximate range and precision for the decimal type are -1.0
X 10-28 to 7.9 X 1028

C# Runtime type : System.Decimal
CSharp declaration : decimal val;
CSharp Initialization : val = 0.12;
CSharp default initialization value : 0.0M

string

The string type represents a string of Unicode characters. string variables are stored any number of alphabetic,

numerical, and special characters .

C# Runtime type : System.String
CSharp declaration : string str;
CSharp Initialization : str = "csharp string";

C# type conversions

Conversion is the process of changing the value of one Type to another. System.Convert class provides a complete set of methods for supported conversions.

In CSharp type conversions are divided into two , Implicit Conversions and Explicit Conversions . Conversions declared as implicit occur automatically, when required and Conversions declared as explicit require a cast to be
called.

```
1:int ctr = 999;
2:long count = ctr;
// implicit conversion from int type to long type
```

From the above statements , first line declare an integer type variable ctr and assigned 999 to it. Second line declare a long type variable count and assign the value of ctr to count. Here the conversion occurred automatically. Because we converted an integer type to long type . This type of conversion is called Implicit
Conversion .

```
1:int ctr = 999;
2:long count = ctr;
// implicit conversion from int type to long type
3:int cnt = (int)count;
// explicit conversion from long type to int type
```

We already saw the Implicit Conversion happened in the second line . The third line again we converted long Type to an integer type . Here we explicitly convert long type to integer (int cnt = (int)count), otherwise the compiler will show compiler error - Error 1 Cannot implicitly convert type 'long' to 'int'. An explicit conversion exists (are you missing a cast?) . This type of conversion is called Explicit Conversion .

The following C# source code shows how to use System.Convert class.

```
private void change()
{
    string str = "true";
      bool flag = System.Convert.ToBoolean(str);
      Console.WriteLine("flg value is " + flag);

}
```

C# Access Modifiers , C# Access Specifiers

Access Modifiers (Access Specifiers) describes as the scope of accessibility of an Object and its members. All C# types and type members have an accessibility level . We can control the scope of the member object of a class using access specifiers. We are using access modifiers for providing security of our applications. When we specify the accessibility of a type or member we have to declare it by using any of the access modifiers provided by CSharp language.

C# provide five access specifiers , they are as follows :

public, private , protected , internal and protected internal .

public :

public is the most common access specifier in C# . It can be access from anywhere, that means there is no restriction on accessibility. The scope of the accessibility is inside class as well as outside. The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private :

The scope of the accessibility is limited only inside the classes or struct in which they are declared. The private members cannot be accessed outside the class and it is the least permissive access level.

protected :

The scope of accessibility is limited within the class or struct and the class derived (Inherited)from this class.

internal :

The internal access modifiers can access within the program that contain its declarations and also access within the same assembly level but not from another assembly.

protected internal :

Protected internal is the same access levels of both protected and internal. It can access anywhere in the same assembly and in the same class also the classes inherited from the same class .

How to use C# if else statements

The conditional statement if.. else in C# is using for check the conditions that we provided in the head of if statement and making decision based on that condition. The conditional statement examining the data using comparison operators as well as logical operators. The else statement is optional , so we can use the statement in
two ways ;

```
if (condition)
statement;
```

```
 if (condition)
        statement;
else
        statement;
```

If the condition is true then the control goes to the body of if block , that is the program will execute the code inside if block.

If the condition is false then the control goes to next level , that is if you provide else block the program will execute the code block of else statement, otherwise the control goes to next line of code.

If you want to check more than one conditions at the same time , you can use else if statement .

```
if (condition)
        statement;
else if (condition)
        statement;
else
        statement;
```

Just take a real-time example - We have a mark list and we want to analyze the grading of each student. In this case we can use if..else conational statements.

Following are the grading rule of the student:

1) If the marks is greater than 80 then the student get higher first class
2) If the marks less than 80 and greater than 60 then the student get first class

3) If the marks less than 60 and greater than 40 then the student get second class

4) If all the above conditions failed and the marks less than 40 then the student is failed.

Now here implementing these conditions in a C# program.

```
1:    if (totalMarks >= 80) {
2:        Console.WriteLine("Got Higher First Class ");
3:    }
4:    else if (totalMarks >= 60) {
5:        Console.WriteLine("Got First Class ");
6:    }
7:    else if (totalMarks >= 40){
8:        Console.WriteLine("Just pass only");
9:    }
10:   else {
11:       Console.WriteLine("Failed");
12:   }
```

Line 1 : Checking the total marks greater than or equal to 80
Line 2 : If total marks greater than 80 show message - "Got Higher First Class "

Line 4 : Checking the total marks greater than or equal to 60

Line 5 : If total marks greater than 60 show message - "Got First Class "

Line 7 : Checking the total marks greater than or equal to 40

Line 8 : If total marks greater than 40 show message - "Just pass only"

Line 10: If those three conditions failed program go to the next coding block .

Line 11: If all fails, it will show message "Failed"

```
private void check()
      {
         int totalMarks = 59;

         if (totalMarks >= 80) {
            Console.WriteLine("Got Higher First Class ");
         }
         else if (totalMarks >= 60) {
            Console.WriteLine("Got First Class ");
         }
         else if (totalMarks >= 40){
            Console.WriteLine("Just pass only");
         }
         else {
            Console.WriteLine("Failed");
         }
      }
```

How to use C# switch case statements

The C# switch statement allows you to choose from many statements based on multiple selections by passing control to one of the case statements within its body. The switch statement executes the case corresponding to the value of the expression . The switch statement can include any number of case instances.

```
 switch (expression)
 {
   case expression:
            //your code here
            jump-statement
   default:
            //your code here
            jump-statement
 }
```

expression : An integral or string type expression.
jump-statement : A jump statement that transfers control out of the case body.

String Switch

C# String Switch Case

The C# language allows you to switch on a string variable. The switch statement compares the String objects in its expression with the expressions associated with each case label as if it were using the String.equals method.

Currently the switch statement is case-sensitive. It would be nice to be able to specify the StringComparison to use for switching on strings. String in switch case make code more readable by removing the multiple if-else-if chained conditions.

```
public void  findStatus(string val)
{
   switch (val)
   {
      case "A+":
        Console.WriteLine("Excellent !!");
         break;
      case "A":
        Console.WriteLine("Very Good  !!");
         break;
      case "B":
        Console.WriteLine("Good  !!");
         break;
      case "C":
        Console.WriteLine("Passed !!");
         break;
      case "D":
        Console.WriteLine("Failed !!");
         break;
      default:
        Console.WriteLine("Out of range !!");
         break;
   }
}
```

C# Switch Case integer

If any of the expression passed to switch case does not match with case statement the control will go to default: statement . If there is no default: statement control will go outside of the switch statement. The following C# program shows how to int values work with Switch..Case statement.

```
private void disp()
{
   int val = 5;
   switch (val)
   {
```

```
            case 1:
              Console.WriteLine("The day is - Sunday");
               break;
            case 2:
              Console.WriteLine("The day is - Monday");
               break;
            case 3:
              Console.WriteLine("The day is - Tuesday");
               break;
            case 4:
              Console.WriteLine("The day is - wednesday");
               break;
            case 5:
              Console.WriteLine("The day is - Thursday");
               break;
            case 6:
              Console.WriteLine("The day is - Friday");
               break;
            case 7:
              Console.WriteLine("The day is - Saturday");
               break;
            default:
              Console.WriteLine("Out of range !!");
               break;
        }
    }
```

How to use C# for loop

There are many situation when you need to execute a block of statements several number of times in your applications. The for loop in C# is useful for iterating over arrays and for sequential processing. This kind of for loop is useful for iterating over arrays and for other applications in which you know in advance how many times you want the loop to iterate. That is the statements within the code block of a for loop will execute a series of statements as long as a specific condition remains true.

Every for loops defines initializer, condition, and iterator sections.

Syntax:

```
for(initialization; condition; step)
code statement
```

initialization : Initialize the value of variable.
condition : Evaluate the condition

step : Step taken for each execution of loop body

The for loop initialize the value before the first step. Then checking the condition against the current value of variable and execute the loop statement and then perform the step taken for each execution of loop body.

```
int count = 4;
for (int i = 1; i < = count; i++)
{
        Console.WriteLine("Current value of i is - " + i);
}
```

The initializer declares and initializes a local loop variable, i, that maintains a count of the iterations of the loop. The loop will execute four times because we set the condition i is less than or equal to count.

```
for (int i = 1; i < = count; i++)
initialization : int i = 1
Initialize the variable i as 1, that is when the loop starts the
value of i is set as 1

condition : i < = count
Set the condition i < =count , that is the loop will execute up to
when the value of i < = 4 (four times)

step : i++
Set the step for each execution of loop block as i++ ( i = i +1)
```

The output of the code as follows :

```
Current value of i is - 1

Current value of i is - 2

Current value of i is - 3

Current value of i is - 4
```

C# Sample For loop program full source code

```
private void repeat()
{
   int count = 4;
   for (int i = 1; i < = count; i++)
   {
      Console.WriteLine("Current value of i is - " + i);
   }
}
```

Infinite Loop

All of the expressions of the for loop statements are optional. A loop becomes infinite loop if a condition never becomes false. You can make an endless loop by leaving the conditional expression empty. The following statement is used to write an infinite loop.

```
for (; ; )
{
  // statements
}
```

Here the loop will execute infinite times because there is no initialization , condition and steps.
break and continue

We can control for loop iteration with the break and continue statements. break terminates iteration and continue skips to the next iteration cycle. The following program shows a simple example to illustrate break and continue
statement.

```
private void disp()
    {
       for (int i = 1; i < = 5; i++)
       {
          if (i == 2) continue;
          if (i == 3) break;
         Console.WriteLine("execute " + i + " times !!");
       }
    }
```

How to use C# foreach loop

The foreach loop in C# executes a block of code on each element in an array or a collection of items. When executing foreach loop it traversing items in a collection or an array . The foreach loop is useful for traversing each items in an array or a collection of items and displayed one by one.

```
foreach(variable type in collection){

  // code block
}
```

variable type : The variable used for collect the item from Collection

collection : Collection of items

```
string[] days = { "Sunday", "Monday", "TuesDay"};
foreach (string day in days)
{
        Console.WriteLine("The day is : " + day);
}
```

The above C# example first declared a string array 'days' and initialize the days in a week to that array. In the foreach loop declare a string 'day' and pull out the values from the array one by one and displayed it.

```
    private void disp()
    {
            string[] days = { "Sunday", "Monday", "TuesDay", "Wednesday", "Thursday", "Friday",
"Saturday" };
        foreach (string day in days)
        {
           Console.WriteLine("The day is : " + day);
        }
```

How to use C# while loop

The while statement continually executes a block of statements until a specified expression evaluates to false . The expression is evaluated each time the loop is encountered and the evaluation result is true, the loop body statements are executed.

```
while(condition)
{
  statement(s);
}
```

c# while loop

Like if statement the while statement evaluates the expression, which must return a boolean value. If the expression evaluates to true, the while statement executes the statement(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

```
int count = 1;
while (count < = 4)
{
 Console.WriteLine("The value of i is : " + count);
 count = count + 1;
}
```

The C# while statement executes a statement or a block of statements until a specified expression evaluates to false . The above program the loop will execute the code block 4 times. A while loop canbe terminated when a break, goto, return, or throw statement transfers control outside the loop. To pass control to the next iteration without exiting the loop, use the continue statement.

while(true)

An empty while-loop with this condition is by definition an infinite loop. You can implement an infinite loop using the while statement as follows:

```
while (true){
  // statements
}
```

```
private void disp()
{
   int count = 1;
   while (count < = 4)
   {
      Console.WriteLine("The value of i is : " + count);
      count = count + 1;
   }
}
```

How to use C# do while loop

The C# while statement executes a statement or a block of statements until a specified expression evaluates to false . In some situation you may want to execute the loop at least one time and then check the condition. In this case you can use do..while loop.

The difference between do..while and while is that do..while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once. From the following example you can understand how do..while loop function.

```
private void disp()
{
   int count = 4;
```

```
        do{
            Console.WriteLine(" Loop Executed ");
            count++;
        }
         while (count < =4);
    }
}
```

How to use C# string Clone

Clone() method creates and returns a copy of string object. The CSharp string Clone() method returns a reference to this instance of string.

Object String.Clone()

Returns:

Object : Return the instance of the String

```
private void disp()
    {

        string str = "Clone() Test";
        string clonedString = null;
        clonedString = (String)str.Clone();
       Console.WriteLine (clonedString);
    }
```

When you run this C# program you will get the same content of the first string "Clone() Test"

How to use C# string Compare

The CSharp String Compare function compares two strings lexicographically . The comparison is based on the Unicode value of each character in the string.

int string.Compare(string str1,string str2)

It returns an Integer indication lexical relationship between the two comprehends

Parameters:

string str1 : Parameter String

string str2 : Parameter String

Returns:

Integer : returns less than zero, zero or greater than zero.

Less than zero : str1 is less than str2

zero : str1 is equal to str2

Greater than zero : str1 is greater than str2

```
 private void disp()
       {
          string str1 = null;
          string str2 = null;

          str1 = "csharp";
          str2 = "CSharp";

          int result = 0;

          result = string.Compare(str1, str2);
         Console.WriteLine(result.ToString());

          result = string.Compare(str1, str2, true);
         Console.WriteLine(result.ToString());
       }
```

How to use C# string Concat

Concat in CSharp String Class Concatenates the two specified string and create a new string.

string concat(string str1,string str2)

String Concat method returns a new String

Parameters:

String str1 : Parameter String

String str2 : Parameter String

Returns:

String : A new String return with str1 Concat with str2

```
private void disp(object sender, EventArgs e)
    {
       string str1 = null;
       string str2 = null;

       str1 = "Concat() ";
       str2 = "Test";
      Console.WriteLine(string.Concat(str1, str2));
    }
```

How to use C# string Contains

The CSharp Contains method returns true if and only if this string contains the specified sequence of char values.

bool string.containe(string str)

Parameters:

String str - input String for search

Returns:

Boolean - Yes/No

If the str Contains in the String then it returns true

If the str does not Contains in the String it returns False

For ex: "This is a Test".Contains("is") return True

"This is a Test".Contains("yes") return False

```
private void disp()
    {
       string str = null;
       str = "CSharp TOP 10 BOOKS";
       if (str.Contains("TOP") == true)
       {
         Console.WriteLine("The string Contains() 'TOP' ");
       }
       else
       {
         Console.WriteLine("The String does not Contains() 'TOP'");
       }
    }
```

How to use C# string Copy

CSharp String Copy method is create a new String object with the same content

string string.Copy(string str)

Parameters:

String str : The argument String for Copy method

Returns:

String : Returns a new String as the same content of argument String

```
private void disp()
                {
                        string str1 = null;
                        string str2 = null;

                        str1 = "CSharp Copy() test";
                        str2 = string.Copy(str1);

                        Console.WriteLine(str2);
                }
```

How to use C# string CopyTo

CSharp string CopyTo method Copies a specified number of characters from a specified position in this instance to a specified position in an array of characters.

```
  void string.CopyTo(int sourceIndex,char[] destination,
  int destinationindex,int count)
```

Parameters:

int sourceIndex : The starting position of the source String

char[] destination : The character Array

int destinationindex : Array element in the destination

int count : The number of characters to destination

```
private void disp()
      {
         string str1 = "CopyTo() sample";
```

```
        char[] chrs = new char[6];
        str1.CopyTo(0, chrs, 0, 6);
                Console.WriteLine(chrs[0].ToString() + chrs[1].ToString() + chrs[2].ToString()+
chrs[3].ToString()  + chrs[4].ToString() + chrs[5].ToString());
    }
```

How to use C# string EndsWith

EndsWith in C# string Class check if the Parameter String EndsWith the Specified String

bool string.EndsWith(string suffix)

Parameters:

suffix - The passing String for it EndsWith

Returns:

Boolean - Yes/No

If the String EndsWith the Parameter String it returns True

If the String doesn't EndsWith the Parameter String it return False

For ex : "This is a Test".EndsWith("Test") returns True

"This is a Test".EndsWith("is") returns False

```
private void disp()
            {
                string str = null;
                str = "VB.NET TOP 10 BOOKS";
                if (str.EndsWith("BOOKS") == true)
                {
                    Console.WriteLine("The String EndsWith 'BOOKS' ");
                }
                else
                {
                    Console.WriteLine("The String does not EndsWith 'BOOKS'");
                }
            }
```

How to use C# string Equals

C# String Equals function is to check the specified two String Object values are same or not

bool string.Equals(string str1,string str2)

Parameters:

String str1 : The String argument

String str2 : The String argument

Returns:

Boolean : Yes/No

It return the values of the two String Objects are same

For ex :

Str1 = "Equals()"

Str2 = "Equals()"

String.Equals(Str1,Str2) returns True

String.Equals(Str1.ToLower,Str2) returns False

Because the String Objects values are different

```
private void disp()
                {
                        string str1 = "Equals";
                        string str2 = "Equals";

                        if (string.Equals(str1, str2))
                        {
                                Console.WriteLine("Strings are Equal() ");
                        }
                        else
                        {
                                Console.WriteLine("Strings are not Equal() ");
                        }
                }
```

How to use C# string Format

string Format method replace the argument Object into a text equivalent System.String.

string string.format(string format,Object arg0)

Parameters:

String format : The format String

The format String Syntax is like {indexNumber:formatCharacter}

Object arg0 : The object to be formatted.

Returns:

String : The formatted String

Exceptions:

System.ArgumentNullException : The format String is null.

System.FormatException : The format item in format is invalid.

The number indicating an argument to format is less than zero, or greater than or equal to

the number of specified objects to format.

examples :

Currency :

String.Format("{0:c}", 10) will return $10.00

The currency symbol ($) displayed depends on the global locale settings.

Date :

String.Format("Today's date is {0:D}", DateTime.Now)

You will get Today's date like : 01 January 2005

Time :

String.Format("The current time is {0:T}", DateTime.Now)

You will get Current Time Like : 10:10:12

```
 private void disp()
       {
          double dNum = 0;
          dNum = 32.123456789;
         Console.WriteLine("Formated String " + string.Format("{0:n4}", dNum));
       }
```

How to use C# string IndexOf

The IndexOf method in string Class in C# returns the index of the first occurrence of the specified substring.

int string.IndexOf(string str)

Parameters:

str - The parameter string to check its occurrences

Returns:

Integer - If the parameter String occurred as a substring in the specified String

it returns position of the first character of the substring .

If it does not occur as a substring, -1 is returned.

Exceptions:

System.ArgumentNullException: If the Argument is null.

example:

"This is a test".IndexOf("Test") returns 10

"This is a test".IndexOf("vb") returns -1

```
private void disp()
      {
         string str = null;
         str = "CSharp TOP 10 BOOKS";
        Console.WriteLine(str.IndexOf("BOOKS").ToString());
      }
```

How to use C# string Insert

The Insert() function in String Class will insert a String in a specified index in the String instance.

string string.Insert(int ind,string str)

Parameters:

ind - The index of the specified string to be inserted.

str - The string to be inserted.

Returns:

String - The result string.

example:

"This is Test".Insert(8,"Insert ") returns "This is Insert Test"

```
private void disp()
      {
         string str = "This is CSharp Test";
         string insStr = "Insert ";
         string strRes = str.Insert(15, insStr);
        Console.WriteLine(strRes);
      }
```

How to use C# string Split

C# Split() handles splitting upon given string and character delimiters. It returns an array of String containing the substrings delimited by the given System.Char array.

split-string-csharp
If your String contains "dd-mm-yy", split on the "-" character to get an array of: "dd" "mm" "yy".

The String Split method ignores any element of separator whose value is null or the empty string ("").

Syntax :

string[] string.split(string[] separator)

Parameters:

separator - the given delimiter

Returns:

An array of Strings delimited by one or more characters in separator

```
private void disp()
      {
         string str = null;
         string[] strArr = null;
         int count = 0;
         str = "CSharp split test";
         char[] splitchar = { ' ' };
         strArr = str.Split(splitchar);
```

```
        for (count = 0; count <= strArr.Length - 1; count++)
        {
          Console.WriteLine(strArr[count]);
        }
    }
```

Output:

```
CSharp
split
test
```

C# String Split Example

How to split strings using regular expressions

The Regular Expressions Split() methods are almost similar to the String.Split() method, except that Regex.Split() method splits the string at a delimiter determined by a Regular Expression instead of a set of characters.

When using Regular Expressions you should use the following namespace in your project

```
using System.Text.RegularExpressions;

string str = "one\n  \ntwo\n  \nthree\n  \n  \nfour";
string[] result = Regex.Split(str, "\n\\s*");
for (int i = 0; i < result.Length; i++)
  Console.WriteLine(result[i]);
```

Output:

```
one
two
three
four
```

c# String Split by multiple characters delimiter

We can split a string by multiple character delimiter using String.split() method.

```
string input = "one)(two)(three)(four)(five";
string[] result = input.Split(new string[] { ")(" }, StringSplitOptions.None);
foreach (string s in result)
  Console.WriteLine(s);
```

Output:

one
two
three
four
five

Using Regular Expressions for multiple characters

C# String Split by multiple characters delimiter using Regular Expressions

```
string input = "one)(two)(three)(four)(five";
string[] result = Regex.Split(input, @"\)\(");
foreach (string s in result)
   Console.WriteLine(s);
```

Output:

one
two
three
four
five

C# String split New Line

You can split a string on a new line or carriage return using the delimiter "\r\n".

C# String split Carriage Return

```
string test = "One\nTwo\r\nThree\nFour\n";
string[] result = test.Split(new string[] { "\n", "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
foreach (string s in result)
   Console.WriteLine(s);
```

Output:

one
two
three
four
Environment.NewLine

Also you can use Environment.NewLine to remove the new line from a string

```
string test = "One\r\nTwo\r\nThree\r\nFour";
string[] result = test.Split(new string[] { Environment.NewLine }, StringSplitOptions.None);
foreach (string s in result)
```

```
   Console.WriteLine(s);
```
Output:

```
one
two
three
four
```
How to split() a delimited string to a List < String >

You can retrieve the result of a String splt() method to a C# List. The following program convert the String Array to a List.

C# Convert List to String

```
string s = "This is a sentence.";
IList<string> list = new List<string>(s.Split(new string[] { " is " }, StringSplitOptions.None));
foreach (string element in list)
{
        Console.WriteLine(element);
}
```

C# String split White spaces

StringSplitOptions.RemoveEmptyEntries guarantees the return value does not include array elements that contain an empty string. The following C# program shows how to remove all white spaces from string using StringSplitOptions.RemoveEmptyEntries.

```
 private void disp()
      {
         string myStrA = "one two   three    four     five";
         string[] result = myStrA.Split(new char[0], StringSplitOptions.RemoveEmptyEntries);
         foreach (string s in result)
           Console.WriteLine(s);

      }
```

How to use C# string Substring

Substring in C# string Class returns a new string that is a substring of this string. The substring begins at the specified given index and extended up to the given length.

```
string string.substring(int startIndex,int length)
```

Parameters:

startIndex: The index of the start of the substring.

length: The number of characters in the substring.

Returns:

The specified substring.

Exceptions:

System.ArgumentOutOfRangeException : the beginIndex or length less than zero, or the begin index + length not within the specified string

```
 private void disp()
       {
          string str = null;
          string retString = null;
          str = "This is substring test";
          retString = str.Substring(8, 9);
         Console.WriteLine(retString);
       }
```

How to validate a string in C#

TryParse is using for determine whether a string is a valid representation of a specified numeric type or not. TryParse method that is implemented by all primitive numeric types and also by types such as DateTime and IPAddress.

bool int.TryParse(string param , out int result)

Parameters:

param: The parameter string.

result: The result will store in this variable

Returns:

returns True or False

```
 private void disp()
       {
          bool isNumeric;
          int i;
          string str = "100";
          isNumeric = int.TryParse(str, out i);
         Console.WriteLine("The value of i is " + i);
       }
```

How to C# String Null

How to handle null String in C#?

A C# string is an array of characters declared using the string keyword. String objects are immutable, meaning that they cannot be changed once they have been created.

What is C# null ?

The null keyword is a special case for a variable value. The implementation of C# on the CLR represents a null reference by zero bits. When defining a string in a class, dont initialize it to null. Instead, initialize it to the constant string.Empty

What is Empty Strings ?

An empty string is an instance of a System.String object that contains zero characters. You can call methods on empty strings because they are valid System.String objects.

```
string s = "";
```

What is Null Strings ?

A null string does not refer to an instance of a System.String object and any attempt to call a method on a null string results in a NullReferenceException. e.g. given below

```
string str = null;
int len  = str.Length;
```

When run the above code it will throw NullReferenceException.

How to check null String in c#?

You may use the null keyword to check or assign the value of an object.

```
string str =null ;
if (str == null)
{
        Console.WriteLine("String is null");
}
```

In the above code we created a string Object and assigned null and next we check the string is null or not.

IsNullOrEmpty method

IsNullOrEmpty is a convenience method that enables you to simultaneously test whether a String is null or its value is Empty.

```
string str =null;
if (string.IsNullOrEmpty(str))
{
        Console.WriteLine("String is empty or null");
}
```

NullReferenceException

NullReferenceException indicates that you are trying to access member fields, or function types, on an object reference that points to null. That means the reference to an Object which is not initialized.

```
private void disp()
                {
                        string str =null;

                        if (str == null)
                        {
                                Console.WriteLine("String is null");
                        }
                        else
                        {
                                Console.WriteLine("String is not null");
                        }

                        str = "test";

                        if (string.IsNullOrEmpty(str))
                        {
                                Console.WriteLine("String is empty or null");
                        }
                        else
                        {
                                Console.WriteLine("String is not empty or null");
                        }

                }
```