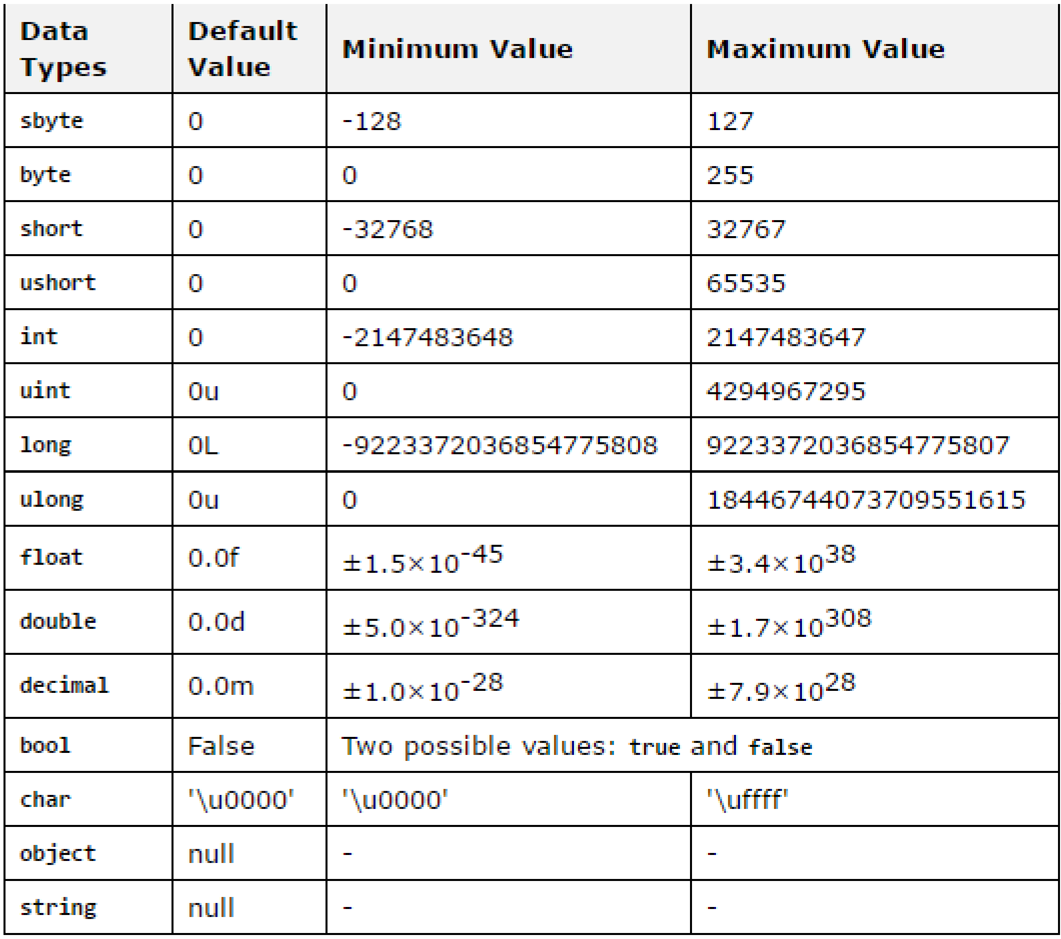
PRIMITIVE DATA TYPES IN C#



	Data Types	Default Value	Minimum Value	Maximum Value	
8-bit	sbyte	0	-128	127	
	byte	0	0	255	
16-bit	short	0	-32768	32767	
	ushort	0	0	65535	
32-bit	int	0	-2147483648	2147483647	
	uint	0u	0	4294967295	
64-bit	long	0L	-9223372036854775808	9223372036854775807	
	ulong	0u	0	18446744073709551615	
	float	0.0f	$\pm 1.5\times 10^{-45}$	$\pm 3.4\times 10^{38}$	**7** digits precision
	double	0.0d	$\pm 5.0\times 10^{-324}$	$\pm 1.7\times 10^{308}$	**15-16** digits precision
128-bit	decimal	0.0m	$\pm 1.0\times 10^{-28}$	$\pm 7.9\times 10^{28}$	**28-29** digits precision
	bool	False	Two possible values: true and false		
	char	'\u0000'	'\u0000'	'\uffff'	
	object	null	-	-	
	string	null	-	-	

=> 1byte = 8 bita

★ Every symbol has an its unique **Unicode** code :

```
char symbol =  'u';
Console.WriteLine("The code of  '{0}' is {1}", symbol, (int)symbol); // The code of 'u' is 117
```

★ **Concatenating** strings :

```
string firstName =  "Bart";
string lastName =  "Simpson";
string fullName = firstName + " " + lastName;
Console.WriteLine("Your full name is {0}", fullName); // Your full name is Bart Simpson
```

★ **Objects** variable taking different type of data :

```
object dataContainer = 7;
Console.WriteLine(dataContainer); // 7
dataContainer = "Seven";
Console.WriteLine(dataContainer); // Seven
```

★ **Assiging** values :

```
int firstValue = 5;
int secondValue;
int thirdValue;

secondValue = firstValue;
Console.WriteLine(secondValue); // 5
thirdValue = firstValue = 3; // assigns is from right to left
```

★ **Initializations** :

```
int num = new int(); // num is = his default value, num = 0
float number = 6.65f; // we initializating float type with 'f' in the end

string greeting = "Hey you!";
string message = greeting; // Hey you!
```

★ **Literals** are :

↳ Boolean -> Return true or false;

```
bool isHuman = true; // true
```

↳ Integer -> Are used for variables of type int, uint, long and ulong;They may have a sign(+,-); They may be in hexadecimal format('0x');

```
int numInHex = -0x10; // -16
int numInDec = -16; // -16
```

↳ Real -> Are used for values of type float, double and decimal;They may consist of digits, a sign and ".";They may be in exponential notation(6.02e+23);

```
float number = 6.65f; // we initializating with 'f' or 'F' in the end for float type, 'd' or 'D'
                         for double, 'm' or 'M' for decimal;
```

↳ Character -> The value may be symbol, the code of the symbol or escaping sequence;

```
char symbol = 'a'; // a
symbol = '\u006F'; // O
symbol = '\n'; // new line
```

↳ String -> Sequence of character literals;

```
string city = "Sofiq";
```

↳ Null -> Wrapper over the primitive types;

```
int? someInt = null; // output is null(print nothing)
someInt = 7; // 7
```

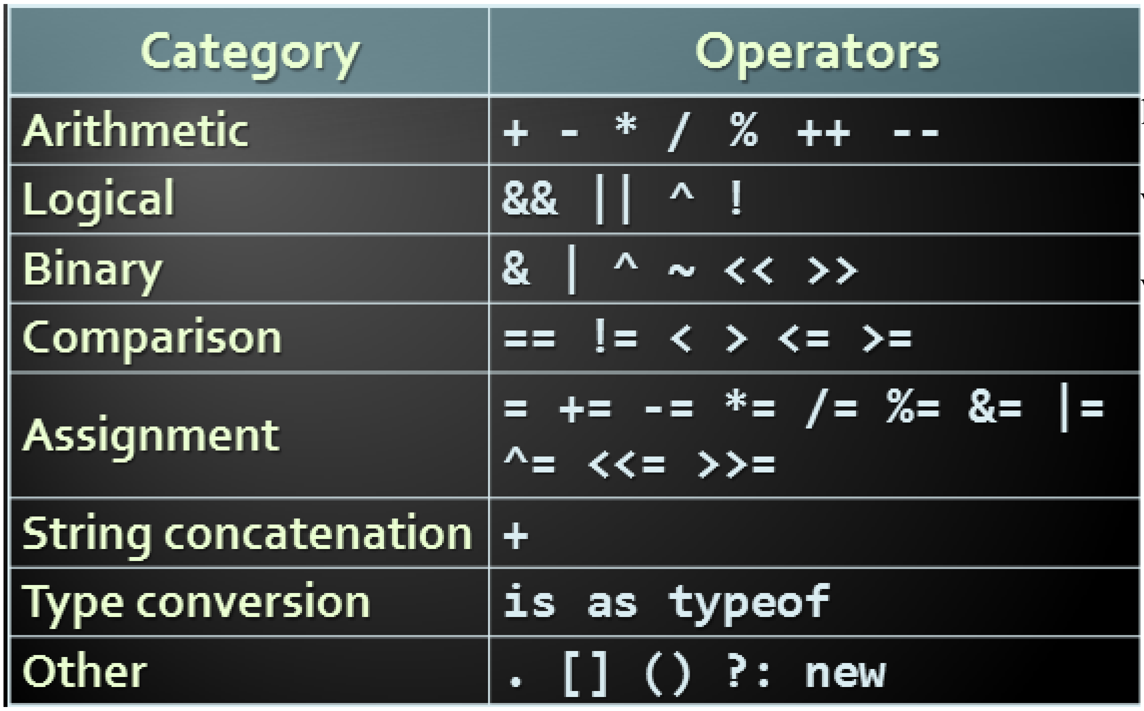
OPERATORS IN C#

★ **Unary** -> Take one operand;

★ **Binary** -> Take two operands;All are left-associative;

★ **Ternary(?:)** -> Take three operands;The assignment operators and conditional(?:) are right-associative

CATEGORIES OF OPERATORS IN C#



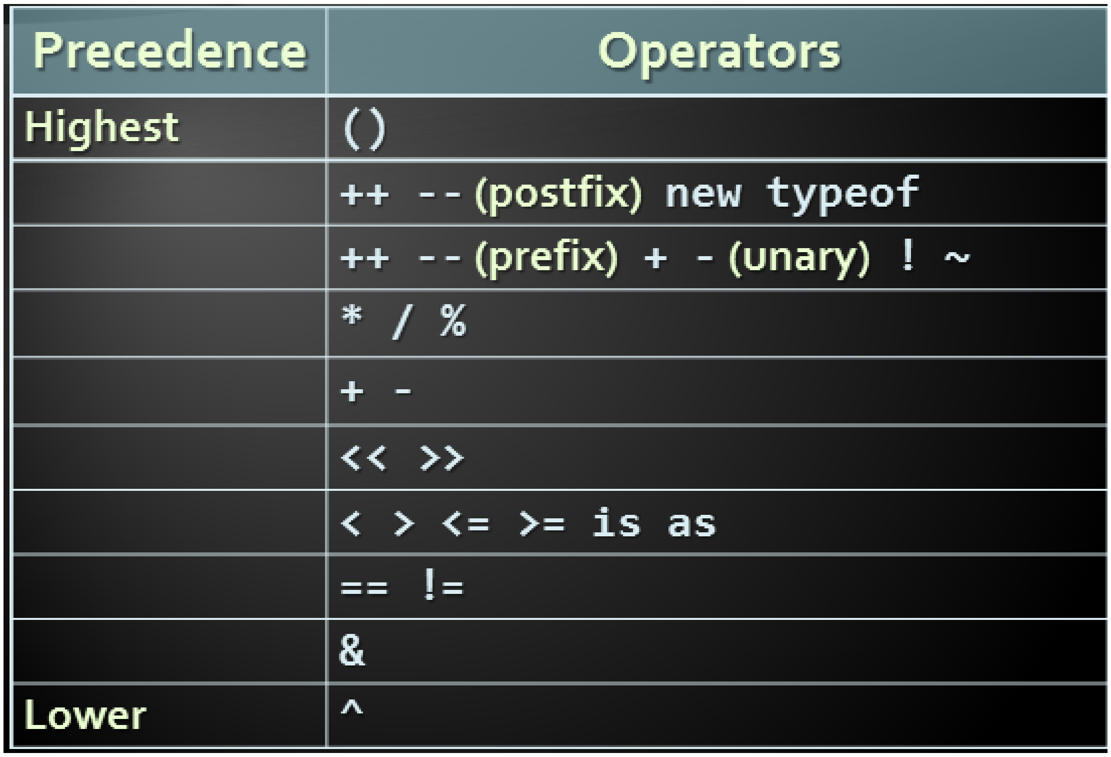
Category	Operators
Arithmetic	+ - * / % ++ --
Logical	&& || ^ !
Binary	& | ^ ~ << >>
Comparison	== != < > <= >=
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
String concatenation	+
Type conversion	is as typeof
Other	. [] () ?: new

like as in math

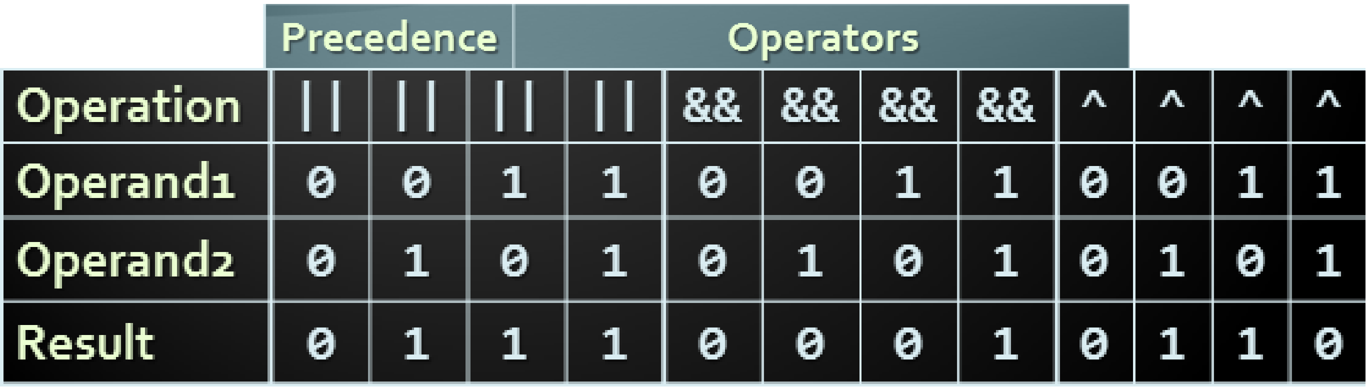
with bool type,return true/false

with binary representacions

OPERATORS PRECEDENCE IN C#



Precedence	Operators
Highest	()
	++ --(postfix) new typeof
	++ --(prefix) + -(unary) ! ~
	* / %
	+ -
	<< >>
	< > <= >= is as
	== !=
	&
Lower	^



	Precedence				Operators							
Operation	||	||	||	||	&&	&&	&&	&&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

★ **Arithmetic** operators -> Are the same as in math;

```
int a = 7;
int b = 3;
Console.WriteLine(a + b); // 10
Console.WriteLine(a - b); // 4
Console.WriteLine(a * b); // 21
```

↳ Division operator "/" returns int whitought rounding;On real numbers returns: real number or Infinity or NaN;

```
Console.WriteLine(7 / 3); // 2

float a = 7.6f;
float b = 3.5f;
Console.WriteLine(a / b); // 2.171428
Console.WriteLine(a / 0.0); // Infinity
Console.WriteLine(-a / 0.0); // -Infinity
Console.WriteLine(0.0 / 0.0); // NaN
```

↳ Remainder operator "%" returns the remainder from devision of integers;

```
int a = 7;
int b = 3;
Console.WriteLine(a % b); // 1
```

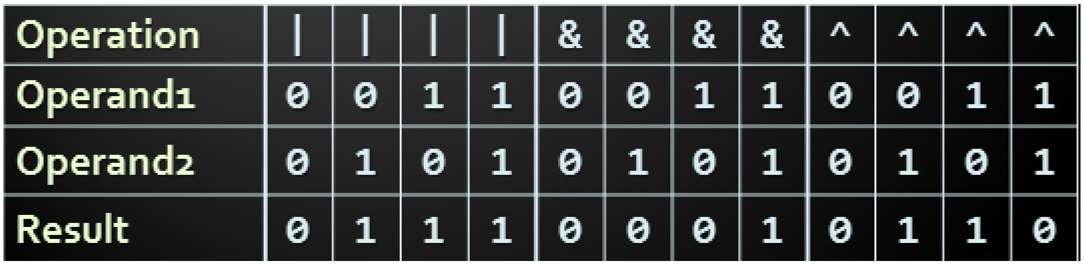
↳ The special addition operator "++" increment a variable;

```
int a = 7;
int b = 3;
Console.WriteLine(a + b++); // 10
Console.WriteLine(a + (++b)); // 12
```

🕮 LOGICAL OPERATORS

```
bool a = true;
bool b = false;

Console.WriteLine(a && b); // False
Console.WriteLine(a || b); // True
Console.WriteLine(a ^ b); // True
Console.WriteLine(!b); // True
Console.WriteLine(b || true); // True
Console.WriteLine(a && true); // True
Console.WriteLine(a || true); // False
Console.WriteLine(!a); // False
```

📖 BITWICE OPERATORS



Operation	|	|	|	|	&	&	&	&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

↳ Are used on integer numbers(byte, sbyte, int, uint, long, ulong);

↳ Are applied bit by bit;

```
ushort a = 3; // 000 000 00 000 000 11
ushort b = 5; // 000 000 00 000 001 01

Console.WriteLine(a | b); // 000 000 00 000 001 11
Console.WriteLine(a & b); // 000 000 00 000 000 01
Console.WriteLine(a ^ b); // 000 000 00 000 001 10
Console.WriteLine(~a & b); // 000 000 00 000 001 00
```

↳ Difference between " = = " and " = " : With operator " = = ", we compare the values of the variables;With " = ", we assign value;

```
int a = 2, b = 4, c= 5;
Console.WriteLine(a = c); // 5;
Console.WriteLine(b = = c); // False
```

☺ TIPS & TRICKS

★ Get the bit at position p in a number n :

```
int p = 5;
int n = 35;                  // 000 000 000 010 001 1
int mask = 1 << p;           // 000 000 000 010 000 0
int nAndMask = n & mask;    // 000 000 000 010 000 0
int bit = nAndMask >> p;    // 000 000 000 000 000 1
Console.WriteLine(bit);     // 1
```

★ **Set the bit at position p to 0 :**

```
int p = 5;
int n = 35;                  // 000 000 000 010 001 1
int mask = ~(1 << p);    // 111 111 111 101 111 1
int result = n & mask;   // 000 000 000 010 001 1
Console.WriteLine(result);  // 3
```

★ **Set the bit at position p to 1 :**

```
int p = 4;
int n = 35;                // 000 000 000 010 001 1
int mask = 1 << p;      // 000 000 000 001 000 0
int result = n | mask;   // 000 000 000 011 001 1
Console.WriteLine(result);  // 51
```

★ **Print a binary number :**

```
Console.WriteLine(Convert.ToString(number, 2).PadLeft(32, '0')); // number is digit you want
                                                                      to convert
```

Other operators

↳ **Conditional (? :) -** has the form **b ? x : y**
If **b** is **true** then the result is **x**, else the result is **y**

```
int a = 4;
int b = 5;
bool isGreatherA =  a > b;
Console.WriteLine(isGreatherA ? "Yes" : "No");  // No
```

↳ **Null-**coalescing operator(**??**) **-** define a default value for both nullable value types and reference types; Returns the left-hand operand if it is not null, else returns the right operand;

```
int? n = null;
int y = n ?? -1;
Console.WriteLine(y); // -1

int? b = 1;
int c = b ?? -1;
Console.WriteLine(c); // 1
```

↳ **Explicit type conversion -** Conversion of a value of one data type to a value of another data type;

```
long p = 10;
int l = (int) p;
Console.WriteLine(l.GetType()); // System.Int32
```

CONSOLE INPUT/ OUTPUT

★ Printing to the Console :

```
string first = "How";
string second = "you";
Console.WriteLine("Hello, world!"); // next printing will start from the new line
Console.Write("{0} are {1}?", how, you); // printing on the same line
```

★ Console Class :

```
Read(); // reads a single character
    int i = Console.Read();
    char ch = (char) i; // cast the int to char
    Console.WriteLine("The code of '{0}' is {1}", ch, i);
ReadLine(); // reads a single line character
     string name = Console.ReadLine(); // print your name
ReadKey(); // reads a combination of keys
     ConsoleKeyInfo key = Console.ReadKey();
     Console.WriteLine();
     Console.WriteLine("character entered:" + key.KeyChar);
     Console.WriteLine("special keys:" + key.Modifiers);
```

★ Read numeral formats :

```
string str = Console.ReadLine(); // read from console
int num = int.Parse(str); // converts a string to int
```

★ Print Custumer formats :

```
Console.WriteLine("{0:0.00}", 1); // 1,00
Console.WriteLine("{0:C2}", 1); // 1,00
Console.WriteLine("{0:#.##}", 0.234); // ,23
Console.WriteLine("{0:#####}", 12345.67); // 12346
Console.WriteLine("{0:(0#) ### ## ##}", 29342525); // (02) 934 25 25
Console.WriteLine("{0:%##}", 0.234); //  #
```

★ Print date formats :

```
DateTime d = new DateTime(2009, 10, 23, 15, 30, 22);
Console.WriteLine("{0:dd/MM/yyyy HH:mm:ss}", d);  //  23.10.2009 15:30:22
Console.WriteLine("{0:d.MM.yy}", d);  // 23.10.09
```

★ TryParse method

```
string str = Console.ReadLine();
int intValue;
bool parseSuccess = Int32.TryParse(str, out intValue);

Console.WriteLine(parseSuccess ?
                    "The square of the number is " + intValue * intValue + "." : "Invalid number!");
```

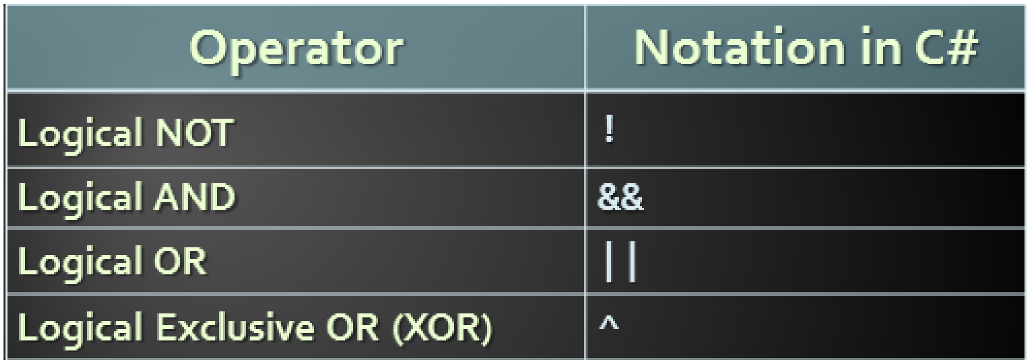
★ **Print special character**

```
using System;
using System.Text;  // needs this library

Console.OutputEncoding = Encoding.UTF8;
Console.WriteLine("България!");
```

★ **Decimal separator**

```
using System;
using System.Threading; // needs this library
using System.Globalization; // needs this library

Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;
Console.WriteLine(3.65);  // 3.65
```

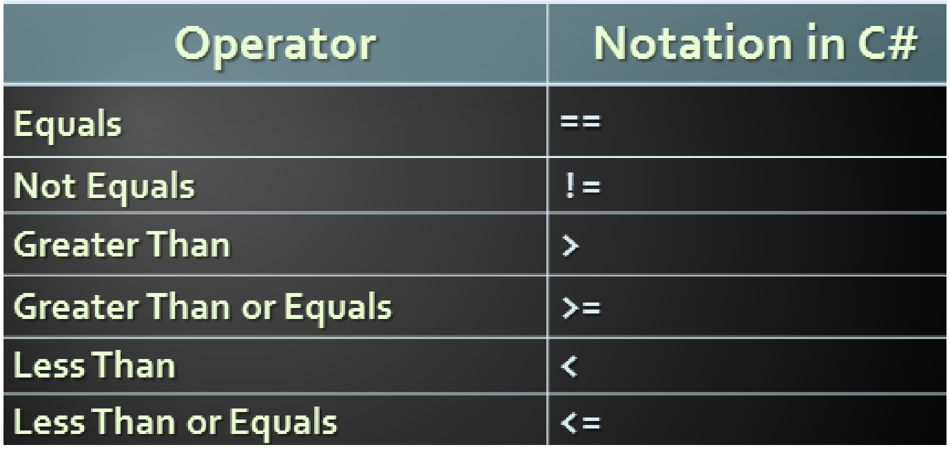
CONDITIONAL STATEMENTS



Operator	Notation in C#
Logical NOT	!
Logical AND	&&
Logical OR	||
Logical Exclusive OR (XOR)	^

★ **De Morgan** laws :

!!A <=> A
!(A || B) <=> !A && !B
!(A && B) <=> !A || !B



Operator	Notation in C#
Equals	==
Not Equals	!=
Greater Than	>
Greater Than or Equals	>=
Less Than	<
Less Than or Equals	<=

★ The **if** statement :

```
int a = 4;
int b = 2;
if (a > b)    // the codition part , if it is true -> the statement is executed, else the statement is
              //  skipped
{
     Console.WriteLine(" '{0}' is greather than '{1}' ", a, b);
}
```

★ The **if- else** statement :

```
int a = 7;
int b = 10;
bool isEqual =  a = = b;

if (isEqual) // condition
{
     Console.WriteLine("Are numbers equal? {0}", isEqual); // if condition is true does this
}
else // if condition isn`t true does this
{
     Console.WriteLine("The numbers aren`t equal.");
}
```

★ **Nested if** statement :

```
int a = 5;
int b = 2;

if (a == b) // condition
{
    Console.WriteLine("Numbers are equals");
}
else
{
     if (a > b) // second condition
     {
        Console.WriteLine(" 'a' is greather than 'b' ");
     }
    else // if nothing of conditions are true
    {
       Console.WriteLine(" 'b' is greather than 'a' ");
    }
}
```

★ Multipule **if-else-if-else** statements :

```
int n = 14;
if ((n % 2 == 0) && n > 0)
{
    Console.WriteLine("The number is even");
}
else if (n % 2 != 0)
{
    Console.WriteLine("The number is odd");
}
else
{
    Console.WriteLine("The number is zero");
}
```

★ **Switch-case** statements :

```
int day = int.Parse(Console.ReadLine());
      switch (day) // switch the paramether 'day' and compare with cases
      {
            case 1 : Console.WriteLine("Monday");break; // if case = 1, print Monday
            case 2 : Console.WriteLine("Tuesday");break;
            case 3 : Console.WriteLine("Wednesday");break;
            case 4 : Console.WriteLine("Thursday");break;
                default : Console.WriteLine("Party day");break; // if case isn`t equal with one of
                                                                  cases, then print default value
      }
```

🕮 LOOPS

★ **While** loop:

```
int i = 0;

      while (i < 10) // condition
      {
         Console.WriteLine("Number : {0}",i); // print digits from 0 - 9
         i++; // refresh number
      }
```

☺ TIPS & TRICKS

★ **Calculate** and print **sum** of the first **N** natural numbers

```
int n = int.Parse(Console.ReadLine());
int number = 1;
int sum= 1;
```

```
while (number  < n)
{
      number++;
       sum + =  number;
}
```

★ Calculate **!n** factorial with **while** loop

```
int n = int.Parse(Console.ReadLine());
int result = 1;

while (true)
{
      if (n = = 1)
      {
           break;
       }
       result *=  n;
       n--;
}
```

★ Calculate **!n** factorial with **BigInteger**

```
using System;
using System.Numerics; // needs this library

int n = int.Parse(Console.ReadLine());
BigInteger result = 1;

do
{
    result *=  n;
    n--;
}
while (n > 0);
```

★ Calculate **!n** factorial with **for** loop

```
int n = int.Parse(Console.ReadLine());
decimal factorial = 1;

for (int i = 1; i  <=  n; i++)
{
      factorial *=  i;
}
```

★ Calculate **N ^ M**

```
int n = int.Parse(Console.ReadLine());
int m = int.Parse(Console.ReadLine());
decimal result = 1;

for (int i = 0; i < m; i++)
{
    result *= n;
}
```

★ **Foreach** loop

```
string[] beers = {"Amstel", "Pirinsko", "ZagorkaMax", "Heineken"};

foreach (string beer in beers)
{
     Console.WriteLine(beer);
}
```

★ **Jump** Statements

```
int outerCounter = 0;

   for (int outer = 0; outer < 10; outer++)
   {
      for (int inner = 0; inner < 10; inner++)
      {
         if (inner % 3 == 0)
            continue; // goes to inner++
          if (outer == 7)
         {
            break; // goes to outerCounter++
          }
         if (inner + outer > 9)
         {
            goto breakOut; // goes to breakOut
          }
      }
      outerCounter++;
   }
breakOut: // label
```