

**Type Casting**

1. Implicit Casting: Converting a smaller type to a larger type.

(char -> int -> long -> float -> double)

**Example:**

int myInt = 12;

double myDouble = myInt; // Automatic casting: int to double

Console.WriteLine(myInt); // Output: 12

Console.WriteLine(myDouble); // Output: 12

1. Explicit Casting: converting a larger type to a smaller type.

(double -> float -> long -> int -> char)

**Example:**

double myDouble = 3.95;

int myInt = (int) myDouble; // Manual casting: double to int

Console.WriteLine(myDouble); // Output: 3.95

Console.WriteLine(myInt); // Output: 3

**User Input**

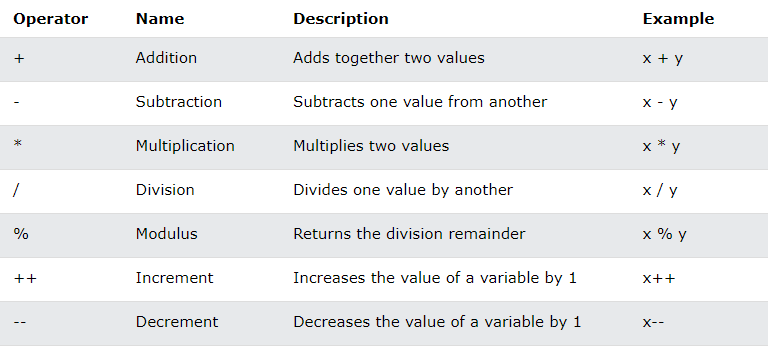
Console.WriteLine("Enter your university name:");

**string name = Console.ReadLine();**

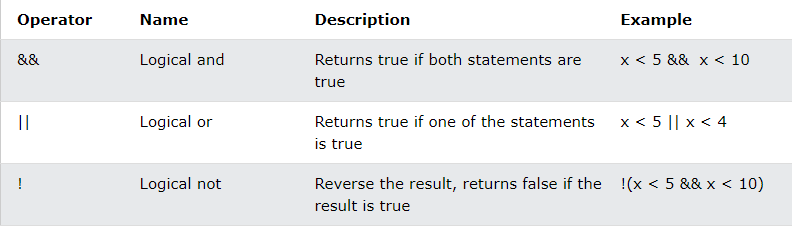
Console.WriteLine("University name is: " + name);

Remember that Console.ReadLine() method returns a string.

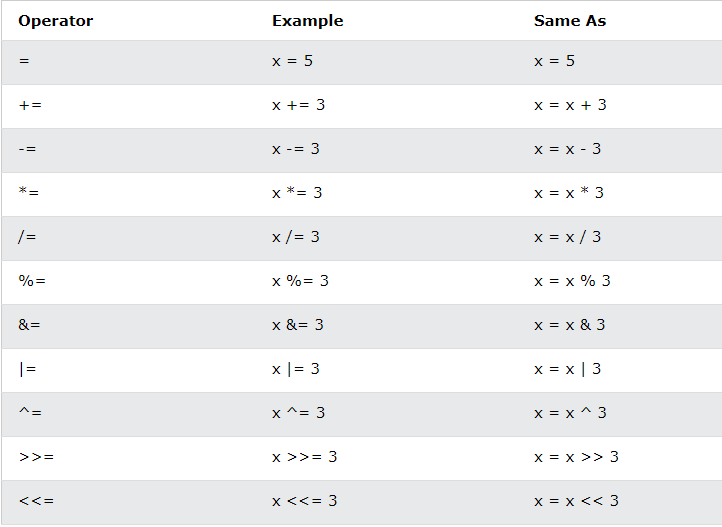
**Arithmetic Operators:**



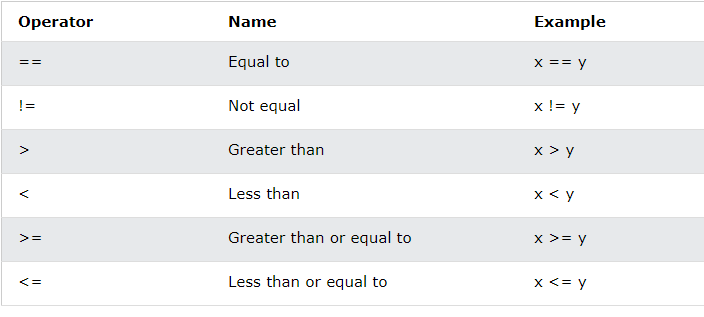
**Logical Operators:**



**Assignment Operators:**



**Comparison Operators:**



**String**

* A string variable contains a collection of characters surrounded by double quotes.
* String length: By using the Length property.

string txt = “Hello World!”

**Console.WriteLine(“Length is: " + txt.Length);**

* Upper-lower conversion: By using ToUpper() and ToLower()

string txt = "Hello World!";

**Console.WriteLine(txt.ToUpper());**

**Console.WriteLine(txt.ToLower());**

* Concatenation: + operator is used.

string firstName = "Anik “;

string lastName = "Islam";

**string name = firstName + lastName;**

Console.WriteLine(name);

* Also used string.Concat() method.

string firstName = "Anik “;

string lastName = "Islam";

**string name = string.Concat(firstName, lastName);**

Console.WriteLine(name);

* Access String: by referring to its index number inside [].

string x = “AIUB";

Console.WriteLine(x[1]);

* Special Characters: by using \”

string x = "AIUB is in \"Dhaka\"";

Console.WriteLine(x);

**Identifiers**

Identifiers are case sensitive.

* + Cannot start with a number
  + Cannot have white space (blank space)
  + Cannot be a reserved keyword

**Naming Conventions**

There are 3 types of naming conventions.

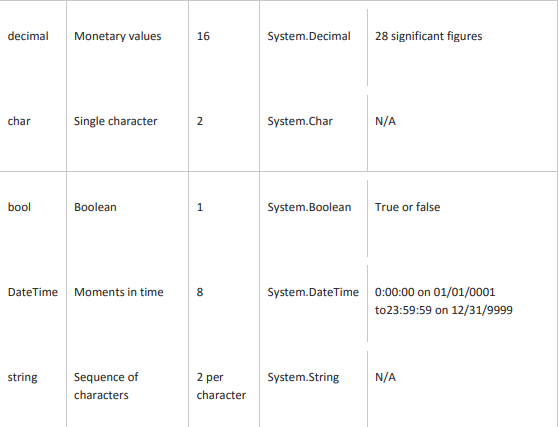
* 1. Camel Case: firstName
  2. Pascal Case: FirstName
  3. Hungarian Notation: Not used in C#

For local variable: Must use Camel Case.

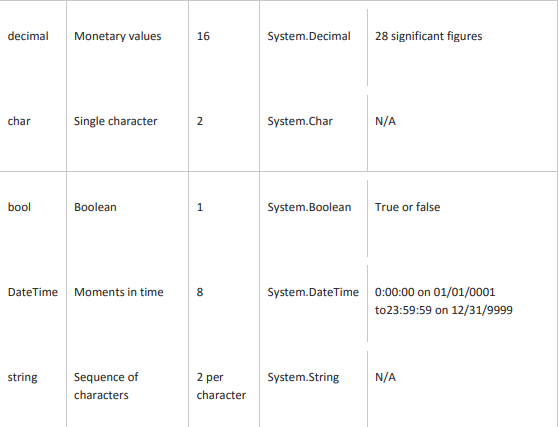
For constant variable: Use Pascal Case.

**Value types data type:**









**Type Casting 2:**

C# also provides some other mechanisms to deal with casting type (from string to other data type).

1. Convert.TodataType() // Convert.ToInt32(Console.ReadLine());
2. dataType.Parse() // Int32.Parse(Console.ReadLine());
3. dataType.TryParse(string, out variableName)

**Example of Convert.TodataType:**

Console.Write("Enter first number:");

int num1 = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter second number:");

int num2 = Convert.ToInt32(Console.ReadLine());

int sum = num1 + num2;

Console.WriteLine("The sum of {0} and {1}: {2}", num1, num2, sum);

**Example of dataType.Parse() and dataType.TryParse():**

int num1 = Int32.Parse(Console.ReadLine()); //4

Console.WriteLine(num1); //4

string a = Console.ReadLine(); //AIUB

Console.WriteLine(Int32.TryParse(a, out int b)); //False

string a = “AIUB”;

Console.WriteLine(Int32.Parse(a)); //Unhandled exception!!

string a = Console.ReadLine(); //4

Console.WriteLine(Int32.TryParse(a, out int b)); //True

Scenario 1

Scenario 2

**Difference between Parse() and TryParse():**

|  |  |
| --- | --- |
| **Parse()** | **TryParse()** |
| 1. Cannot handle exception. | 1. Can handle exception. |
| 1. Only one parameter. | 1. Two parameters are used. |
| 1. Cannot return Boolean value. | 1. Return type: Boolean value (True or False) |

**Constants**

Constants come in three flavors.

* 1. Literals
  2. Symbolic Constants
  3. Enumerations

1. **Literals:** 32 is a literal you cannot change its value.
2. **Symbolic Constants:** Assign a name to a constant value. You declare a symbolic constant using the const keyword and the following syntax:

const type identifier = value;

You must initialize a constant when you declare it, and once initialized it, it can’t be altered. Example-

const int FreezePoint = 32;

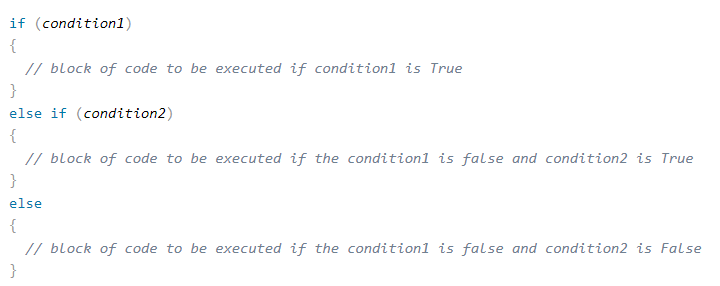
Here, 32 is literal constant and FreezePoint is a symbolic constant of type int.

1. **Enumerations:**

* It is a distinct value type, consisting of a set of named constants. Also known as enumerator list.
* The enum keyword is used to declare an enumerations.
* By default, the first enumeration has the value 0, and the value of each successive enumerator is increased by 1. For example, in the following enumeration, Sat is 0, Sun is 1, Mon is 2, and so on.

enum DayName {Sat, Sun, Mon, Tues, Wed, Thurs, Fri};

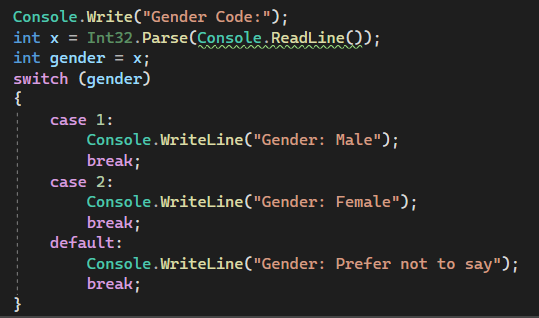
**if, else, else if**



**Switch**

* Switch is used to select one of many code blocks to be executed.
* break keyword, it breaks out of the switch block.
* The default keyword is optional and specifies some code to run if there is no case match.

**EX**



**Repetition**

* + 1. For Loop
    2. While Loop
    3. Do While Loop

**Methods**

* A method is a block of code which only runs when it is called. Example-

class Program {

static void Main(string[] args) {

CallMethod();

}

static void CallMethod() {

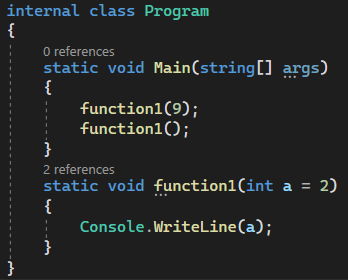
Console.WriteLine("Method is called!");

}

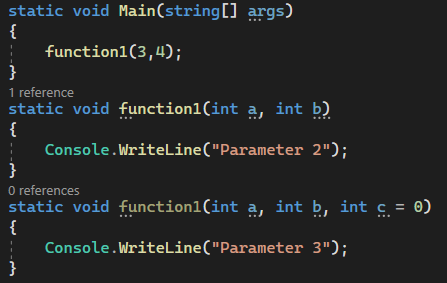
}

**Default Parameter:**

* Created by using the equal (=) sign.
* Call the method without an argument means called the default value.



**Multiple Parameter:**



**Named Parameter:**

static void Main(string[] args)

{

function1(3,c:5);

}

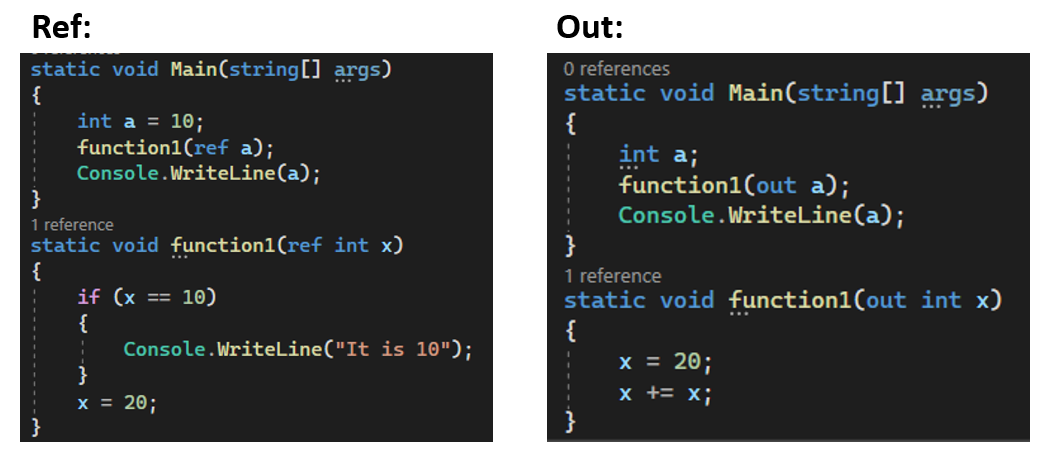
static void function1(int a, int b = 0, int c = 1)

{

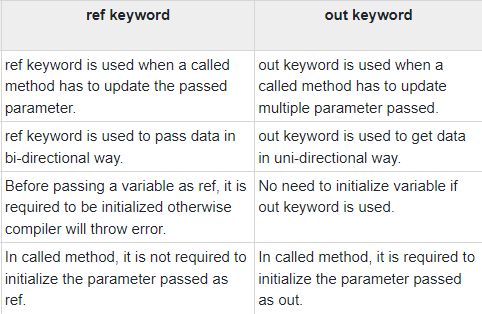
Console.WriteLine("{0} {1} {2}",a,b,c);

}

**Use of Ref and Out Keywords**

****

**Difference between ref & out**



**Struct**

static void Main(string[] args) {

Coordinate point;

point.x = 20;

point.y = 40;

Console.WriteLine(point.x); //output: 20

Console.WriteLine(point.y); //output: 40

}

struct Coordinate{

public int x;

public int y;

}

**Array**

**Introducing Array:**

Array is a sequence of elements, all of which are the same type. You can build simple arrays that have one dimension (a list), two dimensions (a table), three dimensions (a cube), and so on.

* Every element in the array contains a value.
* Arrays are zero-indexed, that is, the first item in the array is element 0.
* The size of an array is the total number of elements that it can contain.

An array has the following properties:

1. An array can be Single-Dimensional, Multidimensional or Jagged.
2. The number of dimensions and the length of each dimension are established when the array instance is created. These values can’t be changed during the lifetime of the instance.
3. The default value of numeric array elements are set to zero, and reference elements are set to null.

**Declaring Array:**

When you declare an array, you specify the type of data that it contains and a name for the array. When you use the new keyword, array is created. At this point, you should specify the size of the array.

Syntax:

*dataType[] arrayName = new dataType [size];*

Example:

int[] arrayName = new int[10];

or

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

or

int[] arrayName = new int[] {1,2,3,4,5,6,7,8};

**Array as objects:**

* In C#, arrays are actually objects.
* Array is the abstract base type of all array types. We can use the properties and other class members, that Array has.

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

int length = number.Length;

Console.WriteLine("Length of the array is {0}",length);

Console.WriteLine(“It is {0} dimension",number.Rank);

**Accessing data in an array:**

You can access data in an array in several ways, such as by specifying the index of a specific element that you require or by iterating through the entire array and returning each element in sequence.

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

int number = oldNumbers[2];

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

int number = oldNumbers[i];

...}

**Single-Dimension array (1D Array):**

You can declare a single dimension array of five integers as shown in the following example.

int[] arrayName = new int[5];

An array that stores string elements can be declared in the same way. Example-

string[] arrayName = new string[6];

**Multidimensional array:**

* An array can have more than one dimension. You can specify up to 32 dimensions, but you will rarely need more than three.
* You declare a multidimensional array variable just as you declare a single-dimensional array, but you separate the dimensions by using commas.

For example, the following declaration creates a 2D array of 4 rows and 2 columns.

int[,] arrayName = new int [4,2];

We can also initialize the array without specifying the numbers of row and column.

int[,] arrayName = {{1,2},{3,4},{5,6},{7,8}}

In order to access elements in a multidimensional array, you must include all indices such as-

int value1 = arrayName[0,0];

int value2 = arrayName[0,1];

**Jagged array:**

* A jagged array is an array whose elements are arrays. The elements of jagged array can be of different dimensions and sizes. A jagged array is sometimes called an ‘array of arrays’.
* Jagged arrays are useful for modeling sparse data structures where you might not always want to allocate memory for every item if it is not going to be used.

The following is the declaration of the jagged array that has three elements and each element is 1D array of integers.

int [][] jaggedArray = new int[3][];

Before using jagged array, its elements must be initialized. We can initialize the elements like this:

jaggedArray[0] = new int[5];

jaggedArray[1] = new int[4];

jaggedArray[2] = new int[2];

It is also possible to use initializers to fill the array elements with values.

jaggedArray[0] = new int[] {1,2,3,4,5};

jaggedArray[1] = new int[]{2,4,6,8};

jaggedArray[2] = new int[]{5,10};

We can also initialize the array upon declaring like this:

int[][] jaggedArray2 = new int[][]

{

new int[] {1,3,5,7,9},

new int[] {2,4,6,8},

new int[] {0,5,10}

};

We can access individual array elements in jagged array like these example:

*//Assign 77 to the second element of first array*

jaggedArray[0][1] = 77;

*//Assign 84 to the second element of second array*

jaggedArray[1][1] = 84

The following is a declaration and initialization of a three elements jagged array and each element is 2D array of integers.

int[][,] jaggedArray2 = new int[3][,]

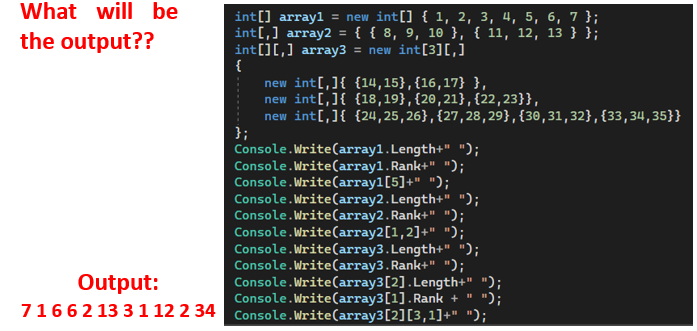
{

new int[,]{ {1,3},{5,7} },

new int[,]{ {0,2},{4,6},{7,8} },

new int[,]{ {1,2},{2,3},{3,4} }

};



**Using foreach with arrays:**

The foreach statement processes elements in the order returned by the array, which is usually from the 0th element to the last. Example-

int[] number = { 1, 2, 3, 4, 5, 6 };

foreach(int i in number)

{

Console.Write(i + " ");

}

**Passing arrays as arguments:**

* Array can be passed as arguments to method parameters.
* Arrays are reference type, the method can change the value of the elements.

int[] theArray = {1,3,5,7};

PrintArray(theArray);

void PrintArray(int[] arr)

{

//method code

}

