- Arrays

Arrays

- **Purpose**: Fixed-size collection of same-type elements
- **Declaration**: `dataType[] arrayName;`
- **Initialization**: `new dataType[size]` or `{ val1, val2 }`
- **Accessing Elements**: `arrayName[index]`
- Length **property**: Number of elements
- **Iteration**: `for` loop, `foreach` loop

Single Dimension Array

- Declare a reference to single dimension array

```
int[] arr;
```

Array Elements Data Type

Variable name

```
char[] arr;
```

Single Dimension Array

- Initialization of array reference
  - ***Explicitly***
    - Array elements auto initialized with default values(0, false, null)
    - **Statically**

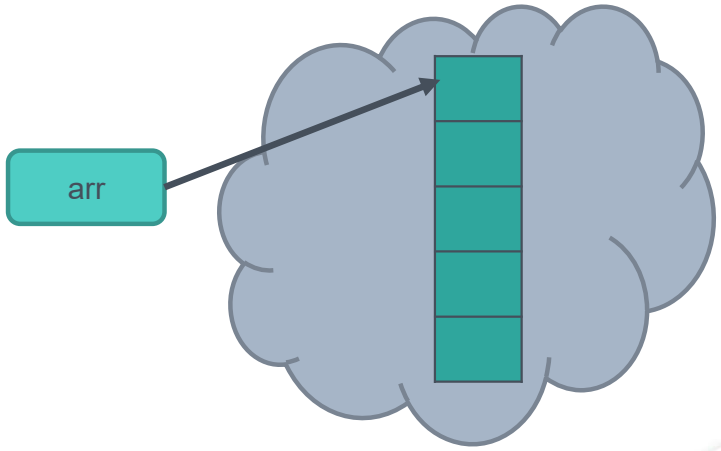
```
int[] arr = new int[5];
```

Array size

    - **Dynamically**

```
arr = new int[size];
```

Array size
(variable)



Single Dimension Array

- Initialization of array reference
  - ***Implicitly*** (Array initializer)


```
int[] arr = new int[] { 10, 50, 3 };
int[] arr = { 10, 50, 3 }; // declaration only
```

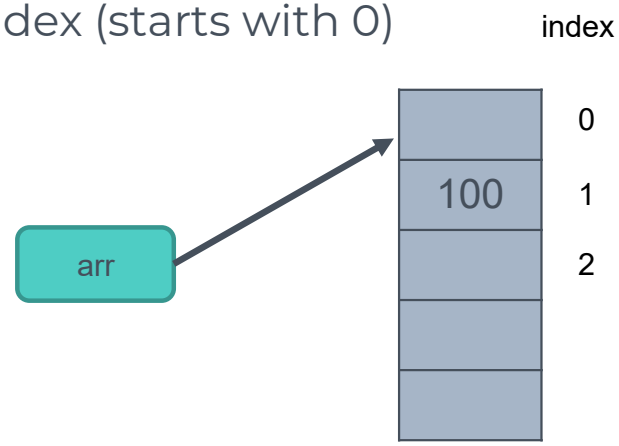
- Array elements automatically initialized with their default values
  - 0,null, false

Single Dimension Array

- Accessing array elements
  - Array elements could be accessed through index (starts with 0)

```
arr[1]=100;
```

index



Single Dimension Array

- **Iteration**
  - `for` Loop

```
for (int i = 0; i < 3; i++)
{
 Console.WriteLine("{0}", arr[i]);
}
```

  - `foreach` loop
    - Used for read only

```
foreach (int x in arr)
{
 Console.WriteLine("{0}", x );
}
```

Arrays: Examples

- Code Example:

```
int[] numbers = { 10, 20, 30 };
Console.WriteLine(numbers[0]); // 10
for (int i = 0; i < numbers.Length; i++) { /* ... */ }
foreach (int num in numbers) { /* ... */ }
```

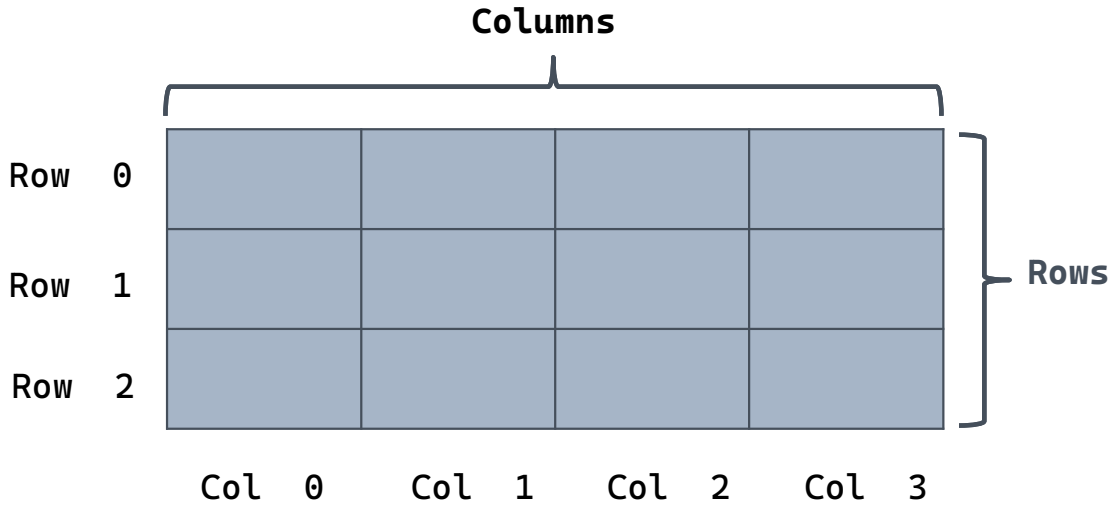
Assignment

- Get sum, average ,max ,min of integers given by the user
  - Let the user determine number of integers
- Calculate the result of one operation Equation
  - Ex: user Input 5*3 ➔ result 15
  - Method used (**`string`**)
    - `Contains`
    - `Split`

```
Equation= 5+3
Result = 8
Equation= 6*5
Result = 30
```

Multi Dimensional Array

- 2-dimension array



Multi Dimensional Array

- Declare reference to multi-dimensional array

```
int[,] arr; // 2-D array
```

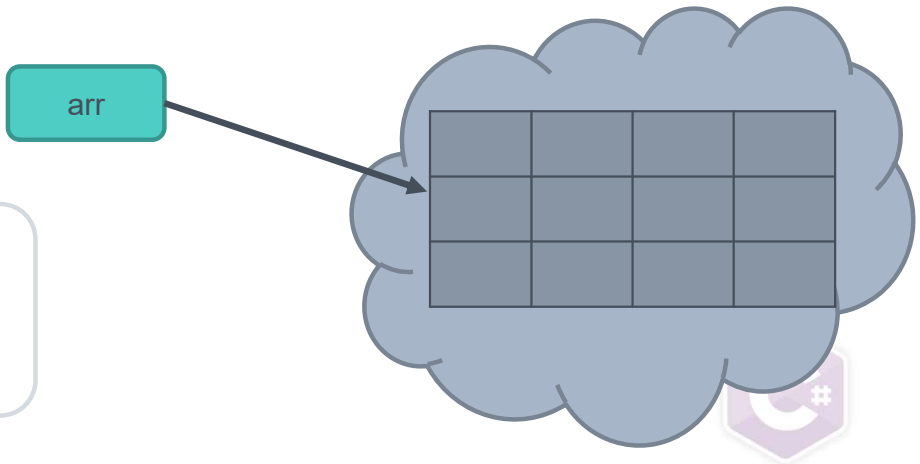
```
int[,,] arr; // 3-D array
```

- Initialization reference
  - *Explicitly*

```
arr = new int[3, 4];
// 3 rows,4 columns
```

  - *Implicitly*

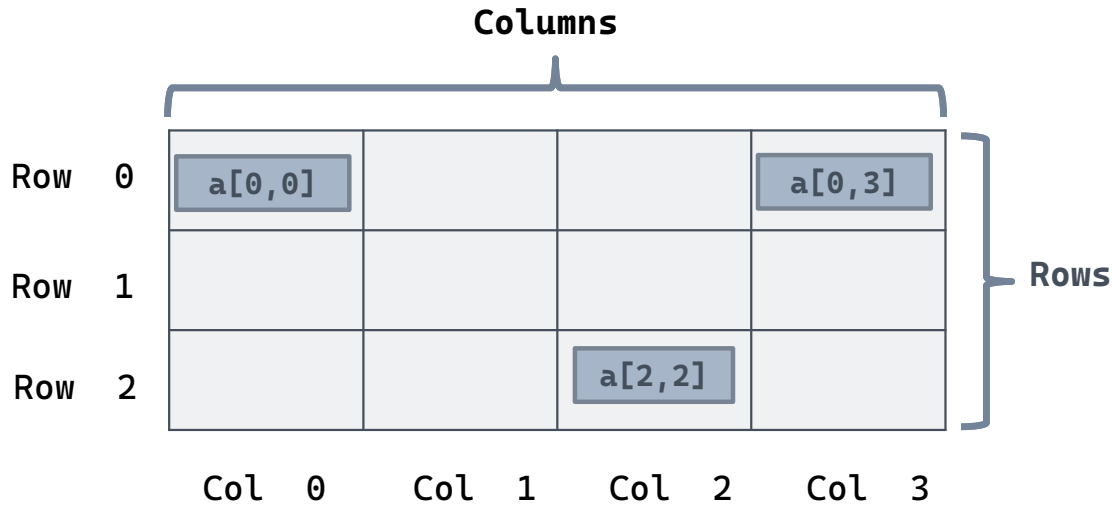
```
int[,] arr = new int[,] {
                {1,2,3},
                {3,4,5}
                }; // 2 rows, 3 columns
```



Multi Dimensional Array

- Access array elements
  - Through 2 indices

```
arr[0,3]=10;
```



Multi Dimensional Array

- Iteration
  - Using 2 nested **for** loop

```
for(int j=0 ; j<3 ; j++)
{
    for(int i=0 ; i<4 ; i++)
    {
    Console.WriteLine (arr[ j ,i]);
    }

}
```

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

Arrays

- Array properties
  - **Length** → number of the array element
- Array Methods
  - Static Methods
    - Sort
    - BinarySearch
    - Reverse
  - instance method
    - GetLength(int dimension)

```
arr = new int[]{5,7,2};
Array.Sort(arr); // Static Method
```

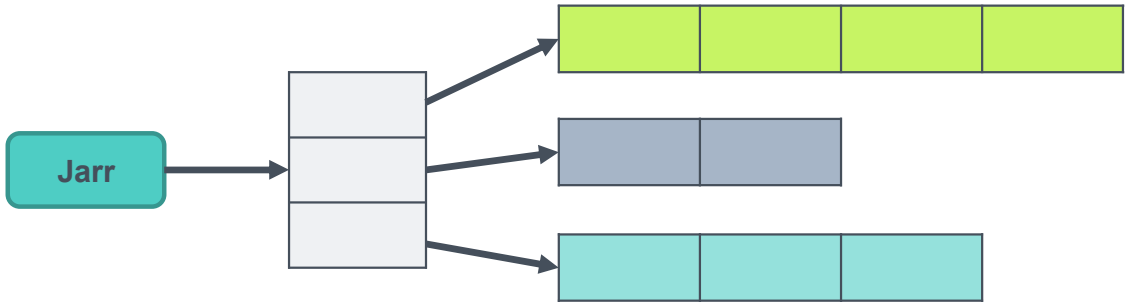
```
int[,] arr = new int[,] { {1,2,3}, {3,4,5} };
arr. GetLength (0); // 2
arr. GetLength (1); // 3
```

Assignment

- Design a program to Get the degree of 3 student with 4 subject from user
  - **calculate**
    - The sum of marks for each student
    - The average for each subject

Jagged Array

- Array of Arrays



- Declare Jagged Array Reference

```
int[][]Jarr ;
```

Jagged Array

- Initialization reference

```
int[][] jarr = new int[3][];
jarr[0] = new int[4] { 1, 2, 3 ,4 };
jarr[1] = new int[2] { 4, 5};
jarr[2] = new int[3] { 10, 15, 20};
```

  - Using array initializer

```
int[][] jArray = new int[][] {
          new int[] { 1, 2, 3 ,4 },
          new int[] { 4, 5},
          new int[] { 10, 15, 20}
            };
```

Assignment

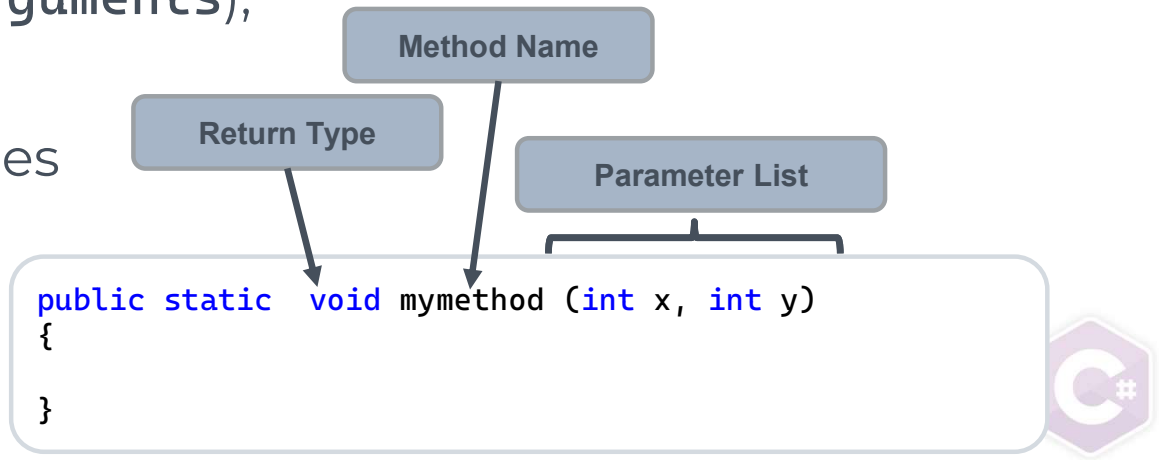
- Design a program that get from user input
  - Number of class room
  - Number of student in each class
  - Mark for each student
- Then calculate the
  - Average mark for each class room

- Methods

Methods: Overview

- **Purpose**: Reusable blocks of code for specific tasks
- **Definition**:
  - Method definition always within a class

  [*access*] [*static*] [*returnType*] `MethodName`([*params*]) { … }
- **Calling**: `MethodName(arguments);`
- `void`: No return value
- `Parameters`: Input values



Methods: Examples

- **No return, no params**: `public void SayHello() { ... }`
- **With params**: `public void Greet(string name) { ... }`
- **With return**: `public int Add(int a, int b) { return a + b; }`
- **Method Overloading**: Same name, different parameters

Methods and memory

- When program starts

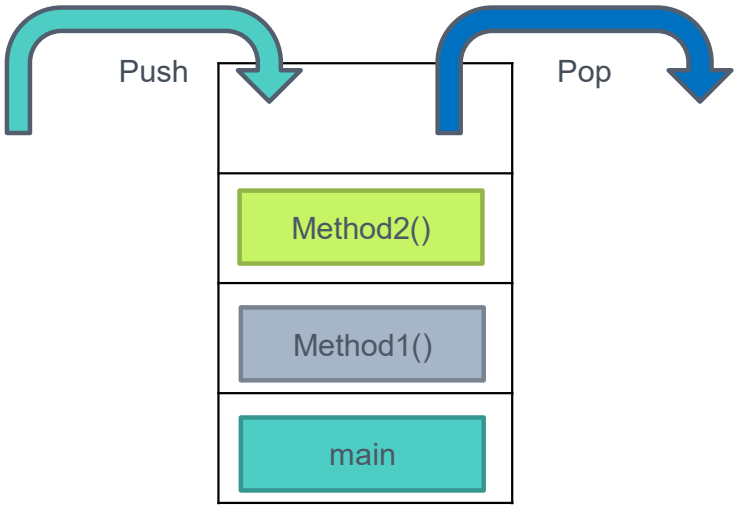
```
static void Main(string[] args)
{
  Console.WriteLine("Main");
  Method1();
  Console.ReadLine();
}
static void Method1()
{
  Console.WriteLine("Method1");
  Method2();
}
static void Method2()
{
  Console.WriteLine("Method2");
}
```

Push

Push

Pop

Pop

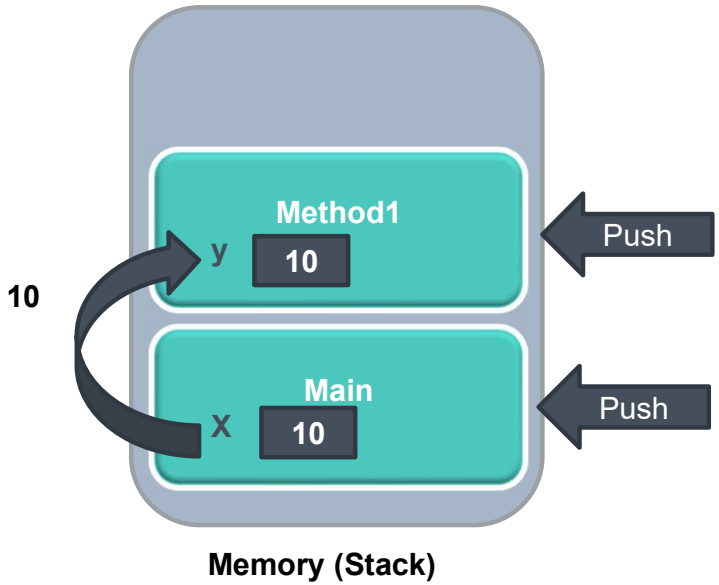


Memory
(Stack)

Method Calling - by value

- ***Value type***
  - Pass value from variable to another

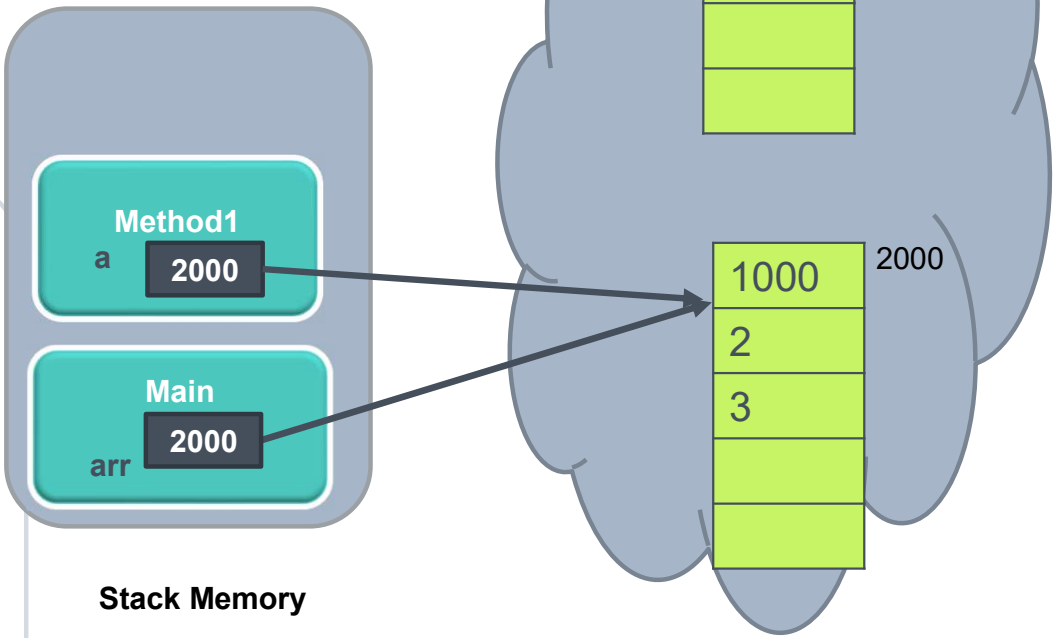
```
static void Main(string[] args)
{
        int x=10;
        Method1(x);
}
static void Method1 (int y)
{
}
```



Method Calling - by value

- ***Reference type***
  - Ex: reference to array

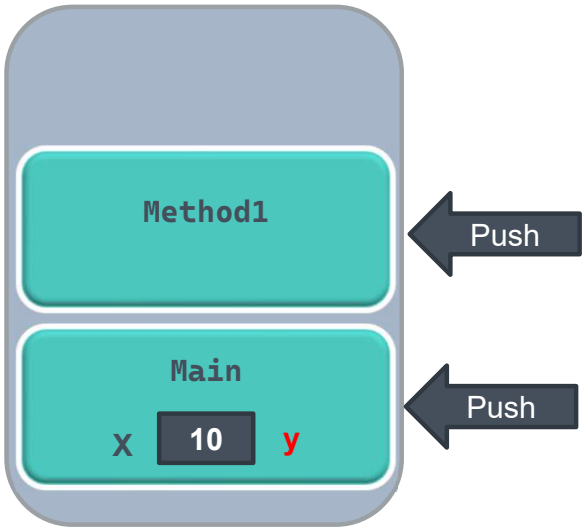
```
static void Main(string[] args)
{
        int[] arr = new int[]{1,2,3} ;
        Method1(arr);
}
static void Method1 (int[] a)
{
        a[0]=1000
        //a=new int[]{5,6}
        //a[1]=200;
}
```



Method Calling - by reference (ref)

Value Type

```
static void Main(string[] args)
{
        int x=10;
        Method1(ref x);
        Console.WriteLine(x);
}
static void Method1 (ref int y)
{
        y=200;
}
```



Memory (Stack)

Method Calling - by reference (**out**)

- **Value Type**
  - Same as **ref** and allow a variable to be passed without initialization
  - Enforce called method to initialize the passed variable
  - Commonly used for return multiple variables from a method

```
static void Main(string[] args)
{
        int x;
        Method1(out x);
        Console.WriteLine(x);
}
static void Method1 (out int y)
{
        y=200;
}
```

Method Calling - by reference (`in`)

- **Value Type**
  - Same as `ref` but modifying passed parameter is **not allowed**
  - Used for performance optimization (passing large structure)

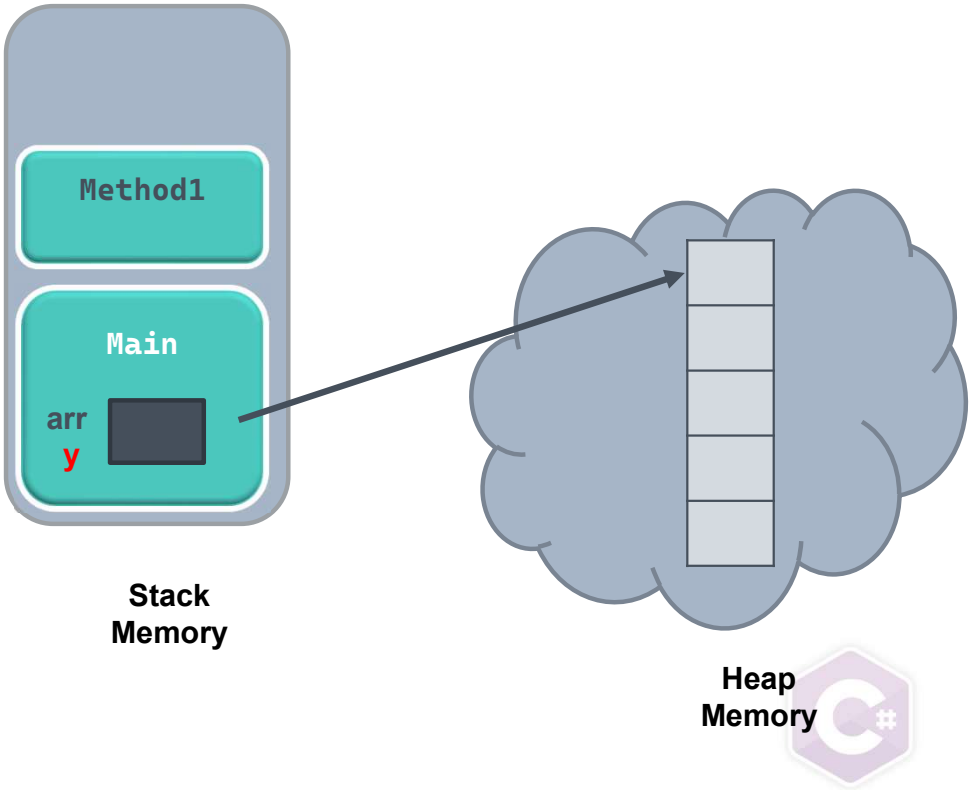
```
static void Main(string[] args)
{
        int x=10;
        Method1(in x);
        Console.WriteLine(x);
}
static void Method1 (in int y)
{
        y=200; //Error
}
```

Method Calling - by reference (`ref`)

- ***Reference type***

```
static void Main(string[] args)
{
        int[] arr=new int[]{1,2,3} ;
        Method1(ref arr);
}
static void Method1 (ref int[] y)
{

}
```



Method Calling

- ***Sequence of passing argument***
  - Which argument is passed first??

```
static void Main(string[] args)
{
    int L = 10;
    Method1(L++, L++, L++);
}
static void Method1(int x, int y, int z)
{
    Console.WriteLine("x ={0}",x);
    Console.WriteLine("y ={0}", y);
    Console.WriteLine("z ={0}", z);
}
```

Recursive method

- A method that contains a self calling
- Must have termination to self calling
  - Ex: power function 2^3 =8

```
int power1(int number, int po)
{
  int result = 1;
  if (po == 0)
      return 1;
  for (int i = 0; i < po; i++)
  {
      result *= number;
  }
  return result;
}
```

```
int power2(int number, int po)
{
    int result = 1;
    if (po == 0)
        return 1;
    result = number * power2(number, po - 1);
    return result;
}
```

Methods

- `params` keyword
  - Allow passing ***reference to array*** to method OR passing the ***elements of Array*** as arguments
  - Definition

```
public static  void mymethod (params int[] x)
{
  x[0]=10;
}
```

  - Calling

```
int []arr=new int[3];
mymethod(arr);

mymethod(10);
mymethod(10,20);
```

Methods

- *Optional Arguments*
  - Set a default value to method parameter
  - It must be the last variable(s) at the right
  - Definition

```
public static void mymethod4(string name,string address=“Giza”)
{
}
```

  - Calling

```
mymethod4("w w w");
mymethod4("w w w", "Cairo");
```

Methods

- *Named Argument*
  - Specify the variable name On method call
  - Definition

```
public static void mymethod4(string name,string address)
{
}
```

  - Calling

```
mymethod4(" ahmed ", "haram street");
mymethod4(address: "haram street",name:"ahmed");
```

Methods

- Passing Parameter to main Method
- Local Method
  - Declare a method within another method
  - Could be called ***only*** within the container method
  - No access modifier used for local method (already private)
  - Overriding is **not** allowed

Assignment

- Menu Program
  - Design Menu program
  - New
    - `NewMethod (…)`
    - Get Employee Data (ID, Name, Salary)
  - Display
    - `DisplayMethod(…)`
    - Display Employee Data
  - Exit
    - `ExitMethod (…)`
    - Exit the program