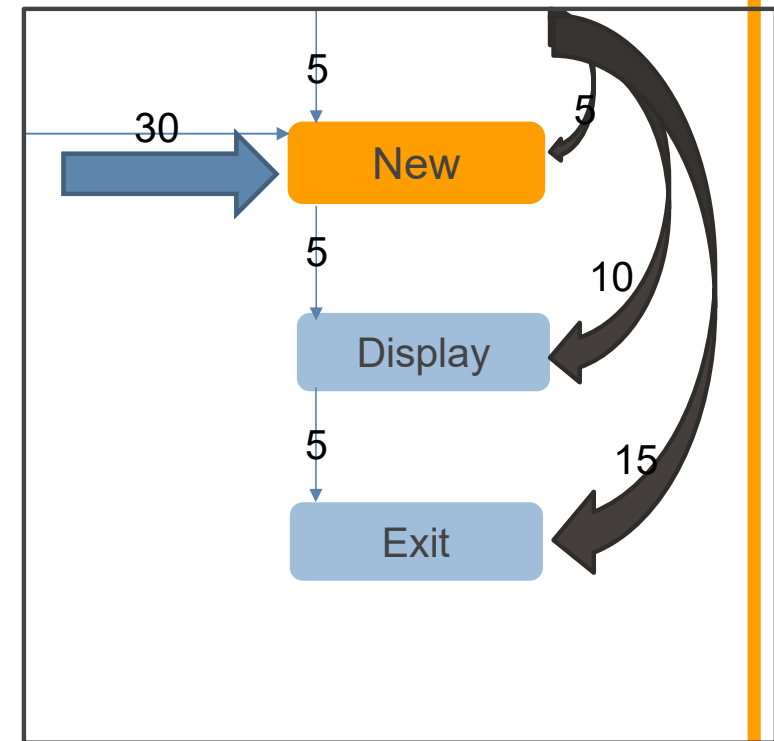




Application on Arrays and Control Statements (Menu Program)

Menu Program

```
String[] Menu = { " New" , "Display " , " Exit " };  
...  
Console.SetCursorPosition(x, y);  
...  
Console.Write(Menu[i]);  
...  
Console.BackgroundColor = ConsoleColor.Blue;  
...  
ConsoleKeyInfo k= Console.ReadKey();
```



Assignment

- ☐ Using Menu Program
 - ☐ New
 - Let user input one employee Data
 - ID
 - Name
 - Salary
 - ☐ Display
 - Display Employee Data
 - ☐ Exit
 - Close the program



Methods

Methods

- A method is a group of statements put together to perform a certain task
- Every C# Application has at least one method *main*
- Method execute through calling

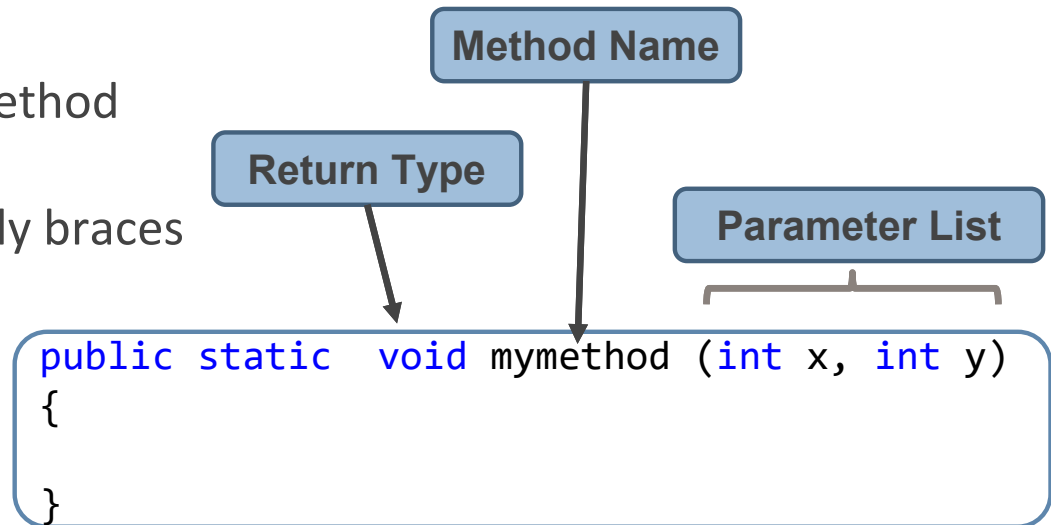
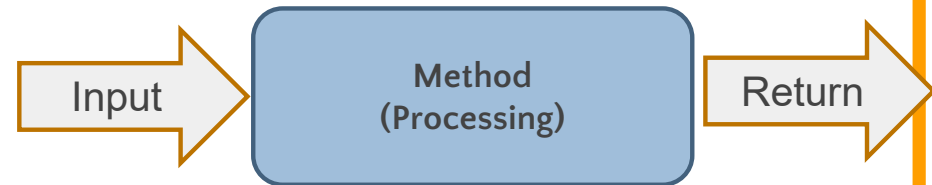
Methods

Method Definition

- Contains
 - Method name
 - Return type
 - Parameters List

Local variables

- Variables declared within method
- Variables passed to method
- Its Scope within method curly braces



Methods

□ Method calling

- Method definition always within a class
- Method calling would be through *Namespace.ClassName.MethodName (static Method)*

```
myspace.myclass.mymethod();
```

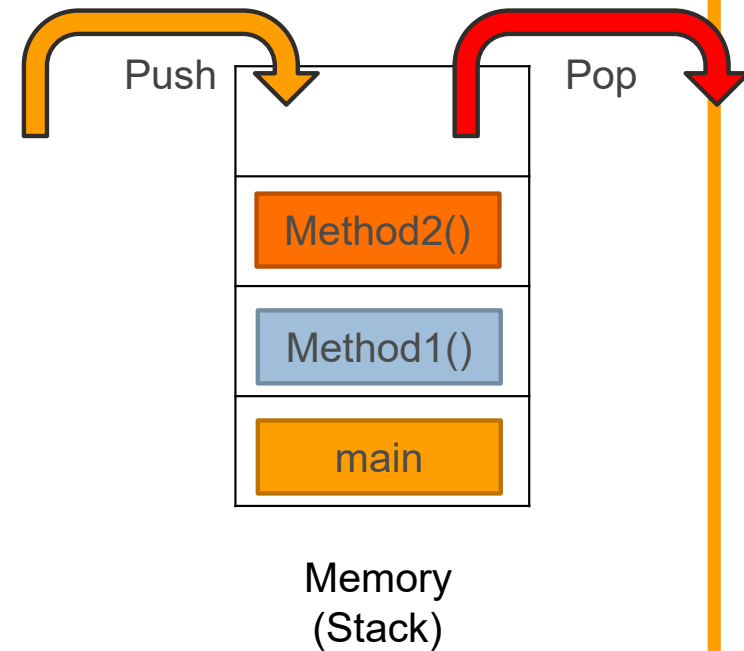
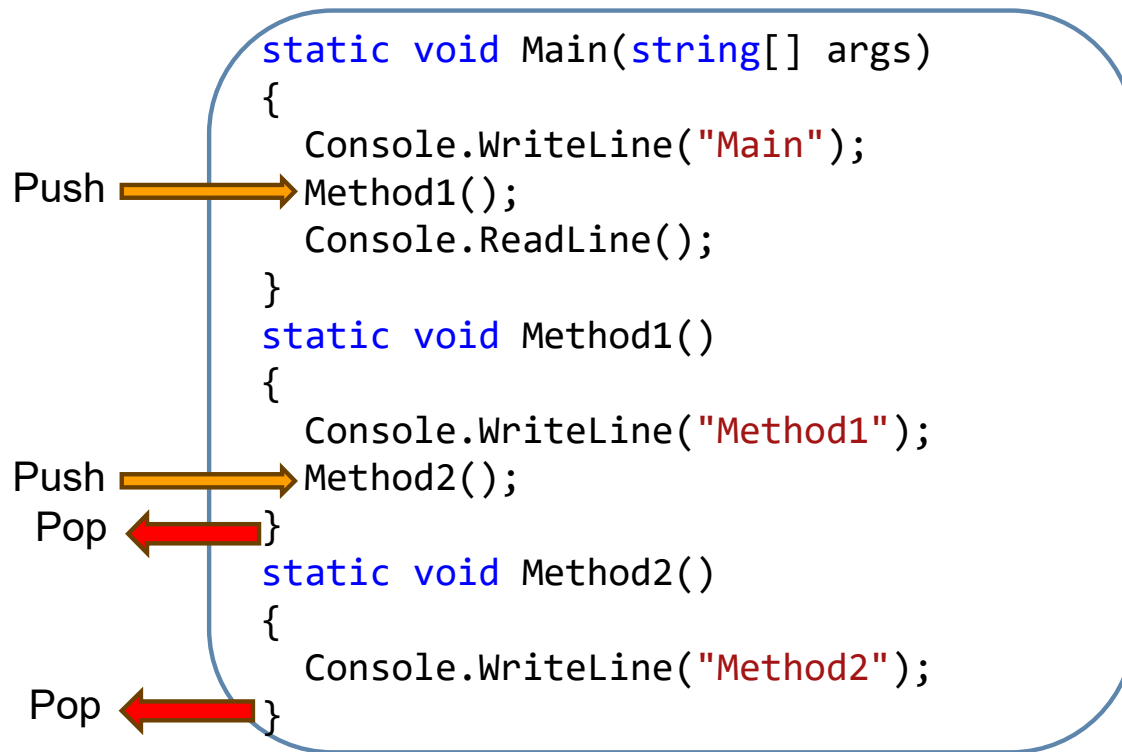
- Method calling would be through *ClassName.MethodName*

```
using myspace;  
...  
myclass.mymethod();
```

```
namespace myspace  
{  
    class myclass // data Type  
    {  
        public static void mymethod()  
        {  
        }  
    }  
}
```

Methods and memory

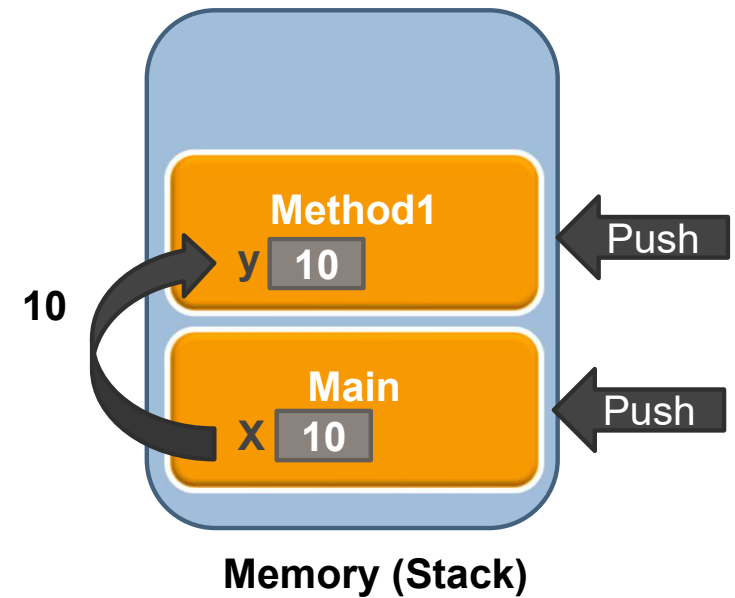
- When program starts
 - Main method pushed to stack



Method Calling

- **Call by value**
 - **Value type**
- Pass value from variable to another
 - Value type Data type

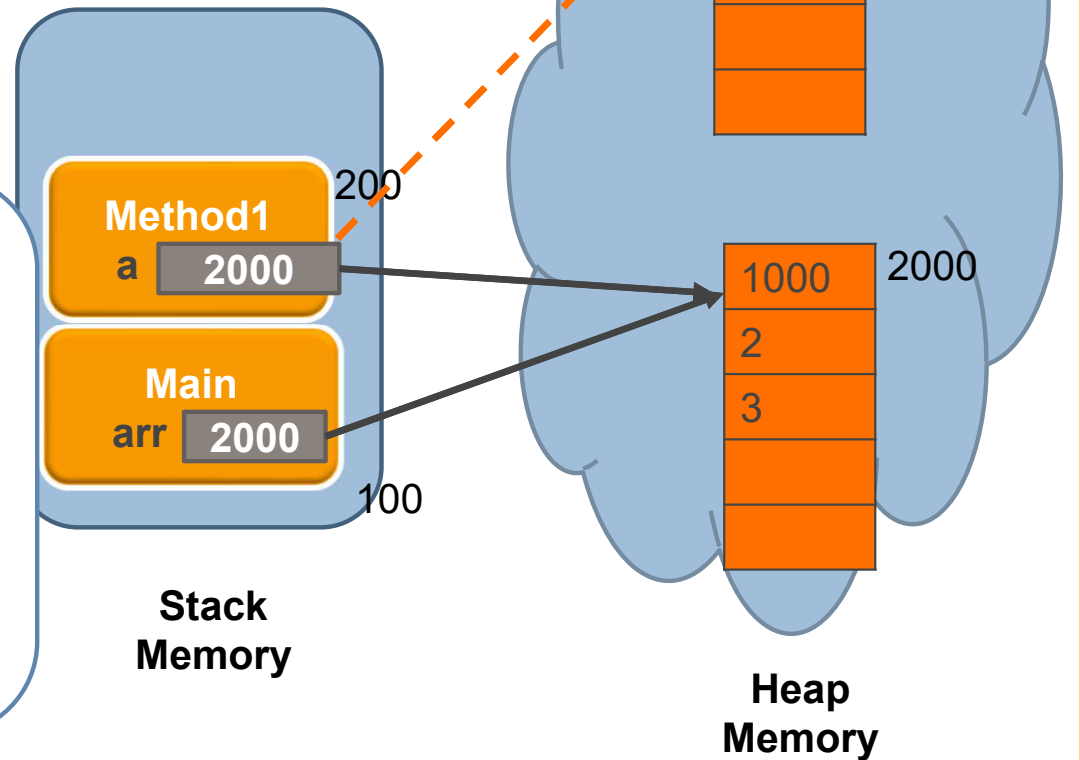
```
static void Main(string[] args)
{
    int x=10;
    Method1(x);
}
static void Method1 (int y)
{
}
```



Method Calling

- **Call 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;
}
```

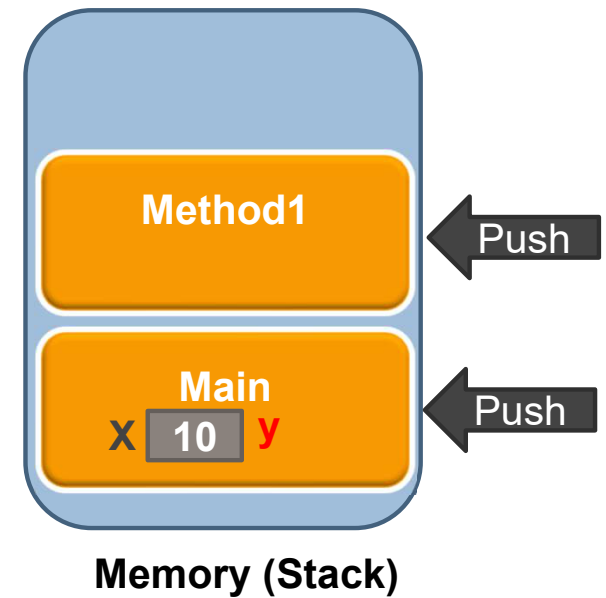


Method Calling

□ *Call 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;
}
```



Method Calling

□ *Call by reference (out)*

- Same as **ref** and allow a variable to be passed without initialization
- Enforce called method to initialize the passed variable

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

Method Calling

□ *Call by reference (in)*

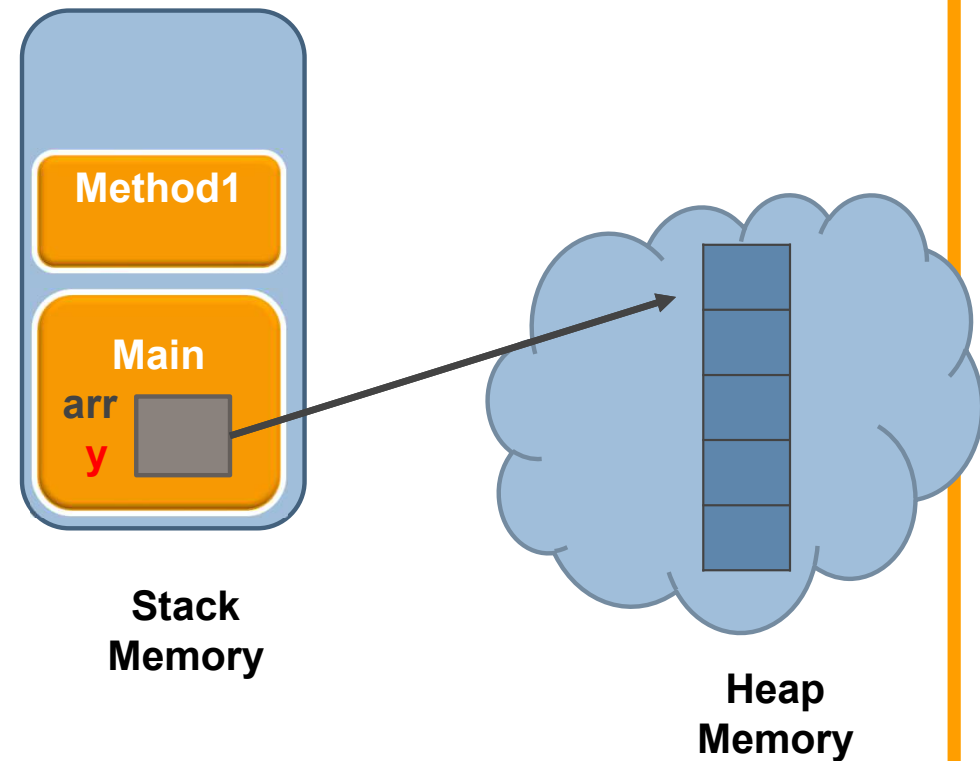
- Same as ref but modifying passed parameter is **not allowed**
- Used for performance optimization (passing large structure)

```
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

- **Call by reference (ref)**
 - **Reference type**

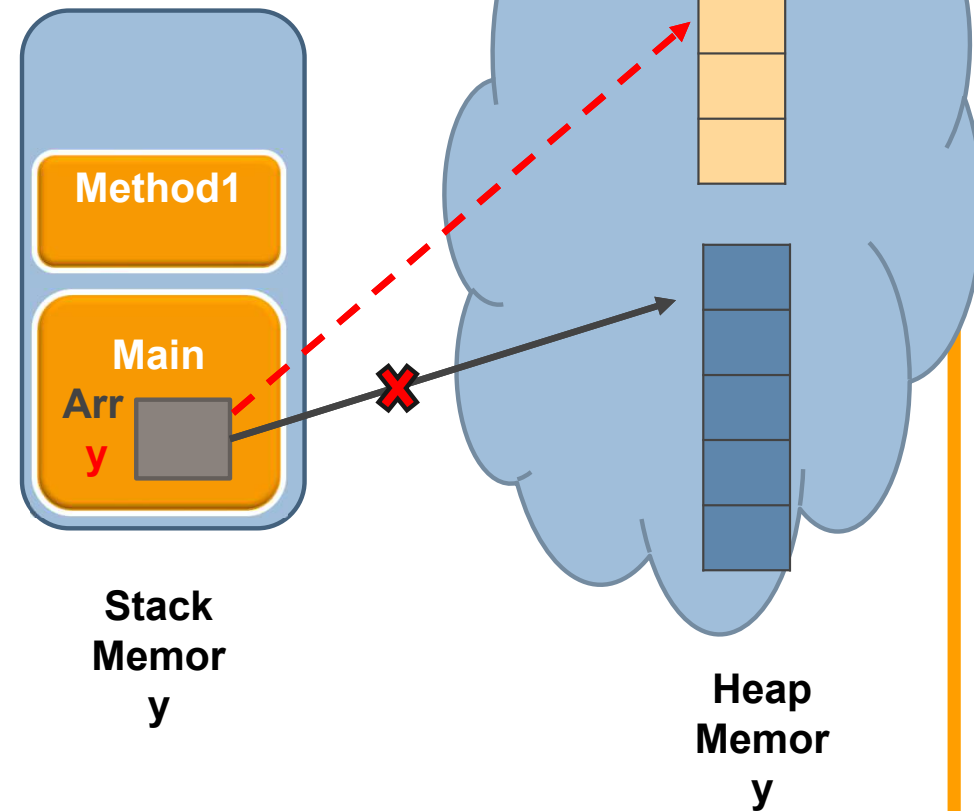
```
static void Main(string[] args)
{
    int[] arr=new int[]{1,2,3} ;
    Method1(ref arr);
}
static void Method1 (ref int[] y)
{
}
```



Method Calling

□ *Special Case*

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



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 $2^3 = 8$

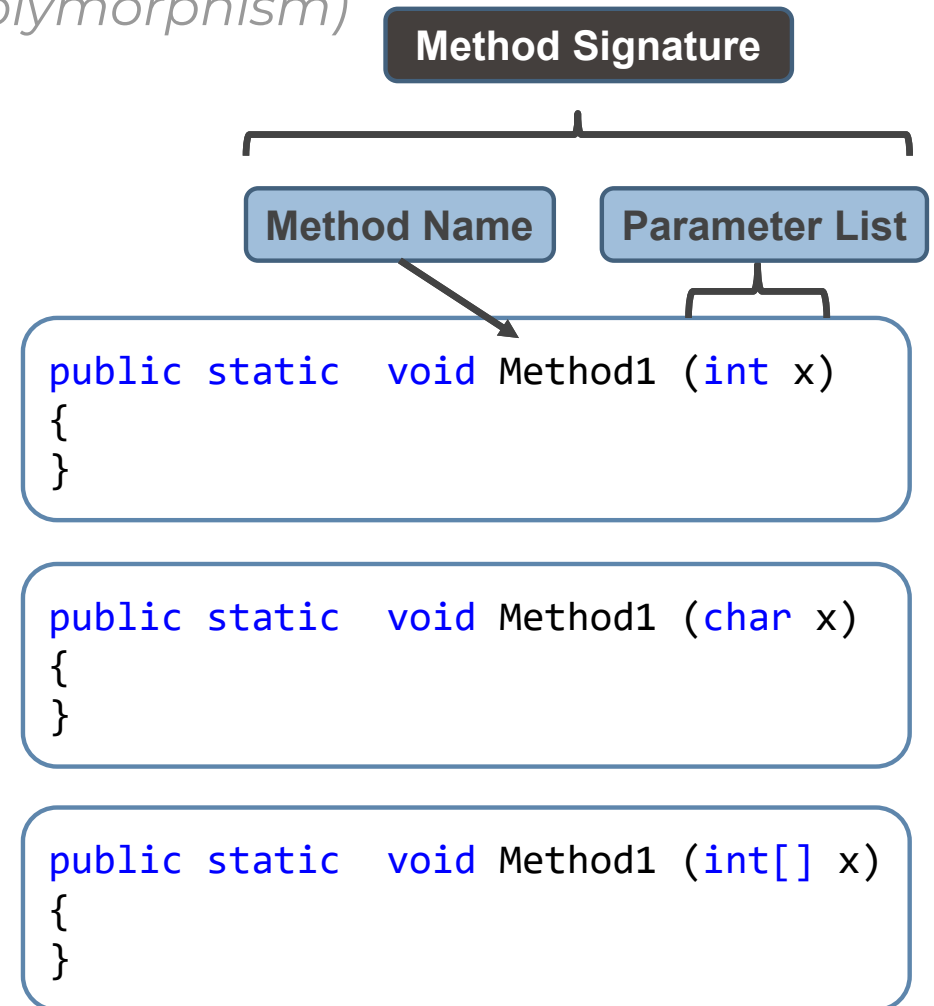
```
int power2(int number,int po)
{
    int result=1;
    if(po==0)
        return 1;
    result=number*power2(number,po-1);
    return result;
}
```

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

Method Overloading

(compile-time polymorphism)

- Define many method with the same but different in parameter (number ,order or type)
- *ref, in , out* and return type not considered in method overloading



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("www");  
mymethod4("www", "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
 - Convert menu program to methods
 - Method for **Display** employee (takes 3 variables : Name , ID, Salary)
 - Method for **Add** new employee (takes 3 variables : Name , ID, Salary)
- Factorial Function (Recursive)