جلسه چهارم سیشارپ
برنامه نویسی پیشرفته
کاردانی نرم افزار ۱

مدرس:
فاطمه دهقانی فیروزآبادی

ارسال آرگومان ها به روش ارجاع

آرگومان‌ها را می‌توان به کمک ارجاع ارسال کرد. این بدان معناست که شما آدرس متغیری را ارسال می‌کنید نه مقدار آن را. ارسال با ارجاع زمانی مفید است که شما بخواهید یک آرگومان که دارای مقدار بزرگی است (مانند یک آبجکت) را ارسال کنید. در این حالت وقتی که آرگومان ارسال شده را در داخل متد اصلاح می‌کنیم مقدار اصلی آرگومان در خارج از متد هم تغییر می‌کند.

در زیر دستورالعمل پایه‌ای تعریف پارامترها که در آنها به جای مقدار از آدرس استفاده شده است نشان داده شده :

```
returnType MethodName(ref datatype param1)
{
   code to execute;
}
```

فراموش نشود که باید از کلمه کلیدی ref استفاده کنید. وقتی یک متد فراخوانی می‌شود و آرگومانها به آنها ارسال می‌شود هم باید از کلمه کلیدی ref استفاده شود.

```
MethodName(ref argument);
```

اجازه دهید که تفاوت بین ارسال با ارجاع و ارسال با مقدار آرگومان را با یک مثال توضیح دهیم.

```
using System;

public class Program
{
    static void ModifyNumberVal(int number)
    {
        number += 10;
        Console.WriteLine("Value of number inside method is {0}.", number);
    }

    static void ModifyNumberRef(ref int number)
    {
        number += 10;
        Console.WriteLine("Value of number inside method is {0}.", number);
    }

    public static void Main()
    {
        int num = 5;

        Console.WriteLine("num = {0}n", num);

        Console.WriteLine("Passing num by value to method ModifyNumberVal() ...");
        ModifyNumberVal(num);
        Console.WriteLine("Value of num after exiting the method is {0}.n", num);

        Console.WriteLine("Passing num by ref to method ModifyNumberRef() ...");
        ModifyNumberRef(ref num);
        Console.WriteLine("Value of num after exiting the method is {0}.n", num);
    }
}
```

```
num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.

Passing num by ref to method ModifyNumberRef() ...
Value of number inside method is 15.
Value of num after exiting the method is 15.
```

در برنامه بالا دو متد که دارای یک هدف یکسان هستند تعریف شده‌اند و آن اضافه کردن عدد 10 به مقداری است که به آنها ارسال می‌شود. اولین متد (خطوط 9-5) دارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متد را صدا می‌زنیم و آرگومانی به آن اختصاص می‌دهیم (خط 24)، کپی آرگومان به پارامتر متد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متد هیچ ارتباطی به پارامتر متد ندارد. سپس مقدار 10 را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم.

برای اثبات اینکه متغیر num هیچ تغییری نکرده است مقدار آن را یکبار دیگر چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است. دومین متد (خطوط 15-11) نیاز به یک مقدار با ارجاع دارد. در این حالت به جای اینکه یک کپی از مقدار به عنوان آرگومان به ان ارسال شود آدرس متغیر به آن ارسال می‌شود. حال پارامتر به مقدار اصلی متغیر که زمان فراخوانی متد به آن ارسال می‌شود دسترسی دارد. وقتی که ما مقدار متغیر پارامتری که شامل آدرس متغیر اصلی است را تغییر می‌دهیم (خط 13) در واقع مقدار متغیر اصلی در خارج از متد را تغییر داده‌ایم. در نهایت مقدار اصلی متغیر را وقتی که از متد خارج شدیم را نمایش می‌دهیم و مشاهده می‌شود که مقدار آن واقعاً تغییر کرده است.

پارامترهای out

پارامترهای out پارامترهایی هستند که متغیرهایی را که مقدار دهی اولیه نشده‌اند را قبول می‌کنند. کلمه کلیدی out زمانی مورد استفاده قرار می‌گیرد که بخواهیم یک متغیر بدون مقدار را به متد ارسال کنیم. متغیر بدون مقدار اولیه، متغیری است که مقداری به آن اختصاص داده نشده است. در این حالت متد یک مقدار به متغیر می‌دهد. ارسال متغیر مقداردهی نشده به متد زمانی مفید است که شما بخواهید از طریق متد متغیر را مقدار دهی کنید. استفاده از کلمه کلیدی out باعث ارسال آرگومان به روش ارجاع می‌شود نه مقدار. به مثال زیر توجه کنید :

```
using System;

public class Program
{
    static void GiveValue(out int number)
    {
        number = 10;
    }

    public static void Main()
    {
        //Uninitialized variable
        int myNumber;

        GiveValue(out myNumber);

        Console.WriteLine("myNumber = {0}", myNumber);
    }
}
```

```
myNumber = 10
```

از کلمه کلیدی out برای پارامترهای متد استفاده شده است بنابراین می‌توانند متغیرهای مقداردهی نشده را قبول کنند. در متد Main ، خط 15 متد را فراخوانی می‌کنیم و قبل از آرگومان کلمه کلیدی out را قرار می‌دهیم. متغیر مقداردهی نشده (myNumber) به متد ارسال می‌شود و در آنجا مقدار 10 به آن اختصاص داده می‌شود (خط 7). مقدار myNumberدر خط 17 نمایش داده می‌شود و مشاهده می‌کنید که مقدارش برابر مقداری است که در داخل متد به آن اختصاص داده شده است (یعنی 10). استفاده از پارامترهای out بدین معنا نیست که شما همیشه نیاز دارید که آرگومانهای مقداردهی نشده را به متد ارسال کنید بلکه آرگومانهایی که شامل مقدار هستند را هم می‌توان به متد ارسال کرد. این کار درحکم استفاده از کلمه کلیدی ref است.

تفاوت ref با out این است که کلمه کلیدی ref به کامپایلر می‌گوید که متغیر مقدار دهی اولیه و بعد به متد ارسال شده است ولی out به کامپایلر می‌گوید که متغیر مقدار دهی اولیه نشده و باید در داخل متد مقدار دهی اولیه شود.

همانطور که قبلاً هم ذکر شد، معمولاً متد از کلمه کلیدی return برای برگشت مقدار استفاده می‌کند. متاسفانه این کلمه کلیدی فقط می‌تواند یک مقدار را برگشت دهد. گاهی اوقات لازم است که یک متد دارای چندین خروجی باشد. اینجاست که از کلمه کلیدی out استفاده می‌شود. به مثال زیر توجه کنید :

```
1  using System;
2
3  class Program
4  {
5      public static void Rectangle(int len, int width, out int area, out int perimeter)
6      {
7          area = len * width;
8          perimeter = 2 * (len + width);
9      }
10
11     static void Main(string[] args)
12     {
13         int area, perimeter;
14
15         Program.Rectangle(5, 4, out area, out perimeter);
16
17         Console.WriteLine("Area of Rectangle is {0}\t", area);
18         Console.WriteLine("Perimeter of Rectangle is {0}\t", perimeter);
19         Console.ReadLine();
20     }
21 }
```

```
Area of Rectangle is 20
Perimeter of Rectangle is 18
```

همانطور که قبلاً هم ذکر شد، معمولاً متد از کلمه کلیدی return برای برگشت مقدار استفاده می‌کند. متاسفانه این کلمه کلیدی فقط می‌تواند یک مقدار را برگشت دهد. گاهی اوقات لازم است که یک متد دارای چندین خروجی باشد. اینجاست که از کلمه کلیدی out استفاده می‌شود. به مثال زیر توجه کنید :

```
1  using System;
2
3  class Program
4  {
5      public static void Rectangle(int len, int width, out int area, out int perimeter)
6      {
7          area = len * width;
8          perimeter = 2 * (len + width);
9      }
10
11     static void Main(string[] args)
12     {
13         int area, perimeter;
14
15         Program.Rectangle(5, 4, out area, out perimeter);
16
17         Console.WriteLine("Area of Rectangle is {0}\t", area);
18         Console.WriteLine("Perimeter of Rectangle is {0}\t", perimeter);
19         Console.ReadLine();
20     }
21 }
```

```
Area of Rectangle is 20
Perimeter of Rectangle is 18
```

متد بالا یعنی Rectangle() قرار است دو خروجی داشته باشد. این دو خروجی محیط (area) و مساحت (perimeter) مستطیل می‌باشند. در نتیجه قبل از آنها و در قسمت پارامترهای متد (خط 5) و هنگام فراخوانی متد کلمه کلیدی out را می‌نویسیم (خط 15).

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به متد ارسال کرد. ابتدا شما باید پارامترهای متد را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```
using System;

public class Program
{
    static void TestArray(int[] numbers)
    {
        foreach (int number in numbers)
        {
            Console.WriteLine(number);
        }
    }

    public static void Main()
    {
        int[] array = { 1, 2, 3, 4, 5 };

        TestArray(array);
    }
}
```

```
1
2
3
4
5
```

مشاهده کردید که به سادگی می‌توان با گذاشتن کروشه بعد از نوع داده‌ای پارامتر یک متد ایجاد کرد که پارامتر آن، آرایه دریافت می‌کند. وقتی متد در خط 17 فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها هم به روش ارجاع به متدها ارسال می‌شوند. در خطوط 7-10 از حلقه foreach برای دسترسی به اجزای اصلی آرایه که به عوان آرگومان به متد ارسال کرده‌ایم استفاده می‌کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```
using System;

public class Program
{
    static void IncrementElements(int[] numbers)
    {
        for (int i = 0; i < numbers.Length; i++)
        {
            numbers[i]++;
        }
    }

    public static void Main()
    {
        int[] array = { 1, 2, 3, 4, 5 };

        IncrementElements(array);

        foreach (int num in array)
        {
            Console.WriteLine(num);
        }
    }
}
```

```
2
3
4
5
6
```

برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. به این نکته توجه کنید که از حلقه foreach نمی‌توان برای افزایش مقادیر آرایه استفاده کنیم چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آنها. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی متد هم اصلاح شده‌اند. راه دیگر برای ارسال آرایه به متد، مقدار دهی مستقیم به متد فراخوانی شده است.

به عنوان مثال :

```
IncrementElements( new int[] { 1, 2, 3, 4, 5 } );
```

در این روش ما آرایه‌ای تعریف نمی‌کنیم بلکه مجموعه‌ای از مقادیر را به پارامتر ارسال می‌کنیم که آنها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه‌ای تعریف نکرده‌ایم نمی‌توانیم در متد Main نتیجه را چاپ کنیم. اگر از چندین پارامتر در متد استفاده می‌کنید همیشه برای هر یک از پارامترهایی که آرایه قبول می‌کنند از یک جفت کروشه استفاده کنید.

به عنوان مثال :

```
void MyMethod(int[] param1, int param2)
{
   //code here
}
```

به پارامترهای متد بالا توجه کنید، پارامتر اول (param1) آرگومانی از جنس آرایه قبول می‌کند ولی پارامتر دوم (param2) یک عدد صحیح. حال اگر پارامتر دوم (param2) هم آرایه قبول می‌کرد باید برای آن هم از کروشه استفاده می‌کردیم:

```
void MyMethod(int[] param1, int[] param2)
{
   //code here
}
```

کلمه کلیدی params

کلمه کلیدی params امکان ارسال تعداد دلخواه پارامترهای همنوع و ذخیره آنها در یک آرایه ساده را فراهم می‌آورد. کد زیر طریقه استفاده از کلمه کلیدی params را نشان می‌دهد :

```
using System;

public class Program
{
    static int CalculateSum(params int[] numbers)
    {
        int total = 0;

        foreach (int number in numbers)
        {
            total += number;
        }

        return total;
    }

    public static void Main()
    {
        Console.WriteLine("1 + 2 + 3 = {0}", CalculateSum(1, 2, 3));

        Console.WriteLine("1 + 2 + 3 + 4 = {0}", CalculateSum(1, 2, 3, 4));

        Console.WriteLine("1 + 2 + 3 + 4 + 5 = {0}", CalculateSum(1, 2, 3, 4, 5));

    }
}
```

```
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
```

از کلمه کلیدی params قبل از نوع داده‌ای آرایه پارامتر استفاده می‌شود (مثال بالا). حال متد را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم. این آرگومانها در داخل یک پارامتر از نوع آرایه ذخیره می‌شوند. با استفاده از حلقه foreach این آرگومانها را جمع و به متد فراخوان برگشت داده می‌شود. وقتی از چندین پارامتر در یک متد استفاده می‌کنید فقط یکی از آنها باید دارای کلمه کلیدی params بوده و همچنین از لحاظ مکانی باید آخرین پارامتر باشد. اگر این پارامتر (پارامتری که دارای کلمه کلیدی params است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر params دار استفاده کنید با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید :

```
void SomeFunction(params int[] x, params int[] y) //ERROR

void SomeFunction(params int[] x, int y, int z) //ERROR

void SomeFunction(int x, int y, params int[] z) //Correct
```

محدوده متغیر

متغیرها در سی شارپ دارای محدوده (scope) هستند. محدوده یک متغیر به شما می‌گوید که در کجای برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متد تعریف می‌شود فقط در داخل بدنه متد قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متد مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند :

```
using System;

public class Program
{
    static void DemonstrateScope()
    {
        int number = 5;

        Console.WriteLine("number inside method DemonstrateScope() = {0}", number);
    }

    public static void Main()
    {
        int number = 10;

        DemonstrateScope();

        Console.WriteLine("number inside the Main method = {0}", number);
    }
}
```

```
number inside method DemonstrateScope() = 5
number inside the Main method = 10
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آنها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متد Main() هیچ ارتباطی به متغیر داخل متد DemonstrateScope() ندارد. وقتی به مبحث کلاسها رسیدیم در این باره بیشتر توضیح خواهیم داد.

پارامترهای اختیاری

پارامترهای اختیاری همانگونه که از اسمشان پیداست اختیاری هستند و می‌توان به آنها آرگومان ارسال کرد یا نه. این پارامترها دارای مقادیر پیشفرضی هستند. اگر به اینگونه پارامترها آرگومانی ارسال نشود از مقادیر پیشفرض استفاده می‌کنند. به مثال زیر توجه کنید :

```
1:  using System;
2:
3:  public class Program
4:  {
5:      static void PrintMessage(string message = "Welcome to Visual C# Tutorials!")
6:      {
7:          Console.WriteLine(message);
8:      }
9:
10:     public static void Main()
11:     {
12:         PrintMessage();
13:
14:         PrintMessage("Learn C# Today!");
15:     }
16: }
```

```
Welcome to Visual C# Tutorials!
Learn C# Today!
```

متد PrintMessage() (خطوط 5-8) یک پارامتر اختیاری دارد. برای تعریف یک پارامتر اختیاری می‌توان به آسانی و با استفاده از علامت = یک مقدار را به یک پارامتر اختصاص داد (مثال بالا خط 5). دو بار متد را فراخوانی می‌کنیم. در اولین فراخوانی (خط 12) ما آرگومانی به متد ارسال نمی‌کنیم بنابراین متد از مقدار پیشفرض (Welcome to Visual C# Tutorials!) استفاده می‌کند. در دومین فراخوانی (خط 14) یک پیغام (آرگومان) به متد ارسال می‌کنیم که جایگزین مقدار پیشفرض پارامتر می‌شود. اگر از چندین پارامتر در متد استفاده می‌کنید همه پارامترهای اختیاری باید در آخر بقیه پارامترها ذکر شوند.

به مثالهای زیر توجه کنید.

```
void SomeMethod(int opt1 = 10, int opt2 = 20, int req1, int req2) //ERROR

void SomeMethod(int req1, int opt1 = 10, int req2, int opt2 = 20) //ERROR

void SomeMethod(int req1, int req2, int opt1 = 10, int opt2 = 20) //Correct
```

وقتی متدهای با چندین پارامتر اختیاری فراخوانی می‌شوند باید به پارامترهایی که از لحاظ مکانی در آخر بقیه پارامترها نیستند مقدار اختصاص داد. به یاد داشته باشید که نمی‌توان برای نادیده گرفتن یک پارامتر به صورت زیر عمل کرد :

```
void SomeMethod(int required1, int optional1 = 10, int optional2 = 20)
{
   //Some Code
}

// ... Code omitted for demonstration

SomeMethod(10, , 100); //Error
```

اگر بخواهید از یک پارامتر اختیاری که در آخر پارامترهای دیگر نیست رد شوید و آن را نادیده بگیرید باید به از نام پارامترها استفاده کنید.

```
SomeMethod(10, optional2: 100);
```

برای استفاده از نام آرگومانها شما به راحتی می‌توانید نام مخصوص پارامتر و بعد از نام علامت کالن (:) و بعد مقدار اختصاص شده به آن را نوشت مانند (optional2: 100) . متد بالا هیچ آرگومانی برای پارامتر اختیاری optional1 ندارد بنابراین این پارامتر از مقدار پیشفرضی که در زمان تعریف متد به آن اختصاص داده شده است استفاده می‌کند.

سربارگذاری متدها

سربارگذاری متدها به شما اجازه می‌دهد که دو متد با نام یکسان تعریف کنید که دارای امضاء و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومانهایی که شما به متد ارسال می‌کنید به صورت خودکار تشخیص می‌دهد که کدام متد را فراخوانی کرده‌اید یا کدام متد مد نظر شماست. امضای یک متد نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید :

```
void MyMethod(int x, double y, string z)
```

که امضای متد بالا

```
MyMethod(int, double, string)
```

به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضای متد نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری متدها آمده است.

```
1  using System;
2
3  public class Program
4  {
5      static void ShowMessage(double number)
6      {
7          Console.WriteLine("Double version of the method was called.");
8      }
9
10     static void ShowMessage(int number)
11     {
12         Console.WriteLine("Integer version of the method was called.");
13     }
14
15     static void Main()
16     {
17         ShowMessage(9.99);
18         ShowMessage(9);
19     }
20 }
```

```
Double version of the method was called.
Integer version of the method was called.
```

در برنامه بالا دو متد با نام مشابه تعریف شده‌اند. اگر سربارگذاری متد توسط سی شارپ پشتیبانی نمی‌شد برنامه زمان زیادی برای انتخاب یک متد از بین متدهایی که فراخوانی می‌شوند لازم داشت. رازی در نوع پارامترهای متد نهفته است. کامپایلر بین دو یا چند متد همنام در صورتی فرق می‌گذارد که پارامترهای متفاوتی داشته باشند. وقتی یک متد را فراخوانی می‌کنیم، متد نوع آرگومانها را تشخیص می‌دهد.

در فراخوانی اول (خط 19) ما یک مقدار double را به متدShowMessage() ارسال کرده‌ایم در نتیجه متد ShowMessage() (خطوط 10-7) که دارای پارامتری از نوع double اجرا می‌شود. در بار دوم که متد فراخوانی می‌شود (خط 20) ما یک مقدار int را به متد ShowMessage() ارسال می‌کنیم متد ShowMessage() (خطوط 15-12) که دارای پارامتری از نوع int است اجرا می‌شود. معنای اصلی سربارگذاری متد همین است که توضیح داده شد.

هدف اصلی از سربارگذاری متدها این است که بتوان چندین متد که وظیفه یکسانی انجام می‌دهند را تعریف کرد تعداد زیادی از متدها در کلاسهای دات نت سربارگذاری می‌شوند مانند متد WriteLine() از کلاس Console. قبلاً مشاهده کردید که این متد می‌تواند یک آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می‌تواند دو یا چند آرگومان قبول کند.

بازگشت (Recursion)

بازگشت فرایندی است که در آن متد مدام خود را فراخوانی می‌کند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود وگرنه برای بی نهایت بار، متد، خود را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت (n!)شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می‌باشد. به فاکتوریل عدد 5 توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

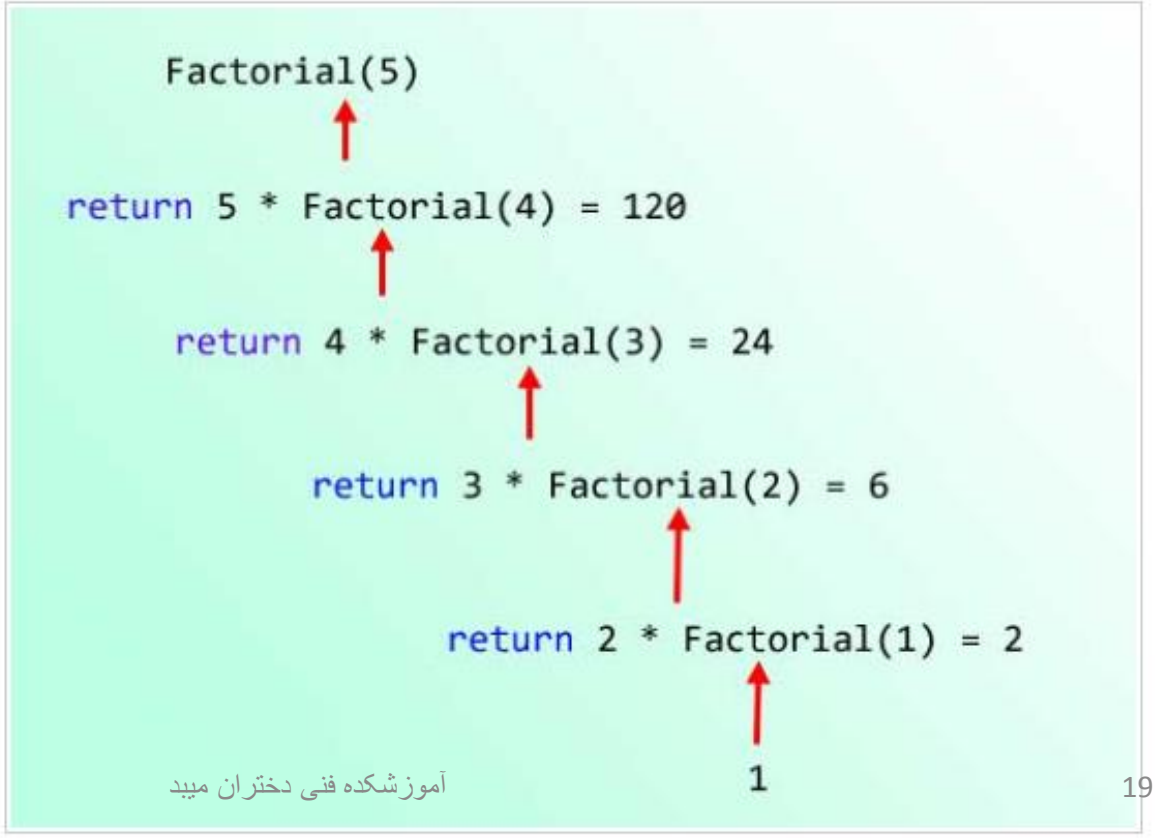
بنابراین برای ساخت یک متد بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بازگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچک‌ترین عدد صحیح مثبت 1 است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```
1  using System;
2
3  public class Program
4  {
5      static long Factorial(int number)
6      {
7          if (number == 1)
8              return 1;
9
10         return number * Factorial(number - 1);
11     }
12
13     public static void Main()
14     {
15         Console.WriteLine(Factorial(5));
16     }
17 }
```

```
120
```

متد مقدار بزرگی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متد یک آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متد یک دستور if می‌نویسیم و در خط 7 می گوییم که اگر آرگومان ارسال شده برابر 1 باشد سپس مقدار 1 را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود.

در خط 10 مقدار جاری متغیر number در عددی یک واحد کمتر از خودش (number – 1) ضرب می‌شود. در این خط متد Factorial خود را فراخوانی می‌کند و آرگومان آن در این خط همان number – 1 است. مثلاً اگر مقدار جاری number 10 باشد یعنی اگر ما بخواهیم فاکتوریل عدد 10 را به دست بیاوریم آرگومان متد Factorial در اولین ضرب 9 خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد 1 برابر نشود. شکل زیر فاکتوریل عدد 5 را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان نوشت.

```
factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
   factorial *= counter;
```

این کد از کد معادل بازگشتی آن آسان‌تر است. از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت حافظه زیادی اشغال می‌کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

آرگومان های خط فرمان (Command Line Arguments)

برای اجرای موفق یک برنامه سی شارپی باید یک متد مهم به نام متد Main() وجود داشته باشد که نقطه آغاز برنامه است. این متد باید به صورت public static تعریف شود. همه ما می دانیم که برای متدها می‌توان آرگومان ارسال کرد، اما برای متد Main(string[] args) چطور؟ جواب مثبت است. شما می‌توانید از طریق دستور خط فرمان ویندوز یا همان CMD آرگومان‌هایی را برای این متد ارسال کنید. برای روشن شدن مطلب یک برنامه کنسول به نام Sample ایجاد کنید، سپس کدهای برنامه را به صورت زیر بنویسید :

```
using System;

namespace Sample
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("First Name is " + args[0]);
            Console.WriteLine("Last  Name is " + args[1]);
            Console.ReadLine();
        }
    }
}
```

برنامه را یک بار اجرا و ذخیره کنید (ممکن است با پیغام خطا مواجه شوید ولی مهم نیست). به پارامتر args توجه کنید. در حقیقت این پارامتر یک آرایه رشته‌ای است که می‌تواند چندین آرگومان از نوع رشته قبول کند. اگر برنامه‌تان را ایجاد کرده و به فایل با پسوند exe. دسترسی داشته باشید می‌توانید پارامترهای رشته‌ای را به متد Main() ارسال کنید. فایل Sample.exe را که در پوشه Debug برنامه‌تان است را به یک درایو یا پوشه مشخص که مسیر گیج کننده‌ای نداشته باشد انتقال دهید. در این مثال ما فایل Sample.exe را مستقیماً در درایو C قرار می‌دهیم. حال CMD ویندوز را اجرا کنید، سپس کدهای زیر (خطوط قرمز) را در داخل CMD نوشته و دکمه Enter را بزنید :

```
Microsoft Windows[Version 6.1.7601]
Copyright(c) 2009 Microsoft Corporation.All rights reserved.

C:\Users\VisualCsharp>cd/

C:\>Sample Steven Clark
First Name is Steven
Last  Name is Clark
```

با نوشتن نام فایل، باعث اجرای آن می‌شویم. بعد از نوشتن نام فایل کلمه Steven و سپس عدد Clark را می‌نویسیم. همانطور که در کد مشاهده می‌کنید ما دو متغیر به نام‌های args[0] و args[1] تعریف کرده‌ایم. این دو متغیر به ترتیب خانه‌های اول و دوم آرایه هستند. کلمه Steven در متغیر رشته‌ای args[0] که اولین عنصر آرایه و کلمه Clark را در متغیر رشته‌ای args[1] که دومین عنصر آرایه است ذخیره و سپس با استفاده از متد WriteLine() آن‌ها را چاپ می‌کنیم. در حقیقت بسیاری از برنامه‌ها از این تکنیک استفاده می‌کنند. شما می‌توانید با ارسال آرگومان‌هایی به متد Main() نحوه اجرای برنامه را تغییر دهید.

تمرین

- برنامه ای بنویسید که تابع فاکتوریل را به صورت بازگشتی و معمولی بنویسید