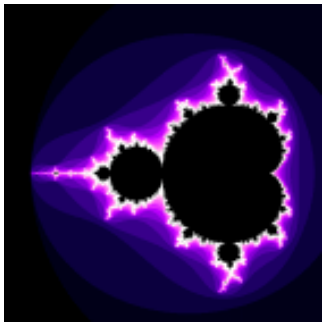
نوشته‌ی مسعود درویشیان

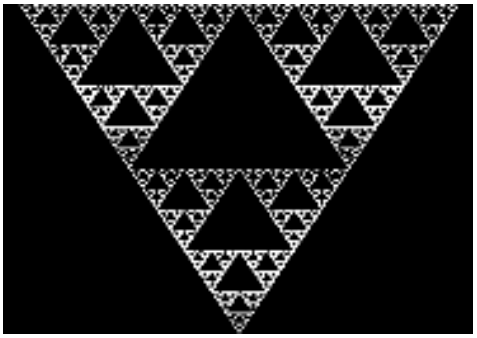
لینک مستقیم این مطلب در وب‌تارگت

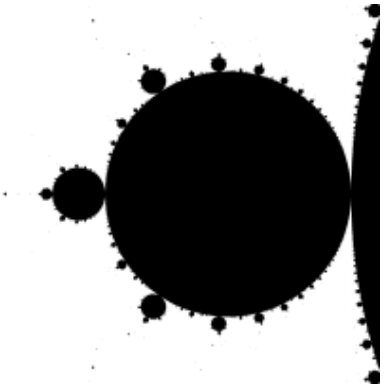
Recursion

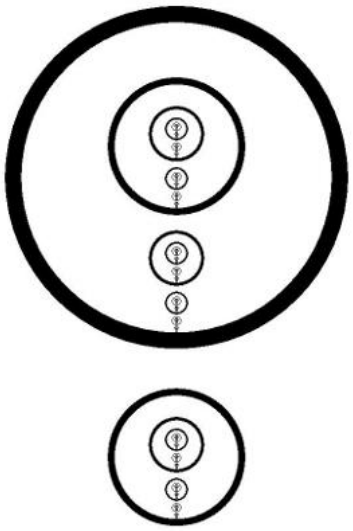
Recurcion پروسه‌ی تکرار آیتم‌ها به‌صورت self-similar (خود متشابه) است. در ریاضیات یک شیء self-similar دقیقاً یا تقریباً شبیه بخشی از خودش است. self-similarity یکی از ویژگی‌های fractal (بَرخال، فرکتال) است. fractal ساختاری هندسی، متشکل از اجزایی است که با بزرگ کردن هر جزء به نسبت معین همان ساختار اولیه به‌دست می‌آید.



به عبارت دیگر fractal ساختاری است که هر جزء آن با کل آن همانند است.







(برای متحرک دیدن تصاویر بالا به وب‌سایت مراجعه کنید)

مثالی دیگر در این مورد، برف‌دانه‌ی کخ است که می‌تواند همراه با بزرگ‌نمایی همواره بدون تغییر باقی بماند.

همان‌طور که گفته شد، recursion پروسه‌ی تکرار آیتم‌ها به‌صورت self-similar است. برای مثال وقتی که سطح دو آینه دقیقاً باهم موازی باشند، عکس‌های تودرتو به‌وجود آمده نوعی recursion نامحدود است. recursion در بعضی از علوم می‌تواند مفهوم و کاربرد خاص خودش را داشته باشد اما بیشترین کاربرد آن در علم ریاضیات و کامپیوتر است.

در ریاضیات، یک مثال کلاسیک از recursion، سری فیبوناچی است:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

```
// fib(0) is 0;
// fib(1) is 1;
// for all integers n > 1: fib(n) is fib(n - 1) + fib(n - 2);
```

یک مثال کلاسیک دیگر در این مورد، فاکتوریل است:

```
// factorial(0) is 1;
// for all integers n > 0: factorial(n) is n * factorail(n - 1);
```

Recursion در علم کامپیوتر

در سی‌شارپ، یک متد می‌تواند خودش را فراخوانی کند (درون خودش، خودش را صدا بزند)، به این پروسه recursion گفته می‌شود و متدی که خودش را صدا زده، recursive است. recursion یک مکانیزم کنترلی قدرتمند است.

در مثال زیر محاسبه‌ی factorial را با روش recursive (بازگشتی) و nonrecursive (غیربازگشتی) می‌بینید:

```
// A simple example of recursion.
using System;
class Factorial
{
    // This is a recursive method.
    public int FactR(int n)
    {
        int result;
        if (n == 0) return 1;
        result = FactR(n - 1) * n;
        return result;
    }
    // This is a nonrecursive method.
    public int FactI(int n)
    {
        int t, result;
        result = 1;
        for (t = 1; t <= n; t++) result *= t;
        return result;
    }
}
class Recursion
{
    static void Main()
    {
        Factorial f = new Factorial();
```

```
        Console.WriteLine("Factorials using recursive method.");
        Console.WriteLine("Factorial of 3 is " + f.FactR(3));
        Console.WriteLine("Factorial of 4 is " + f.FactR(4));
        Console.WriteLine("Factorial of 5 is " + f.FactR(5));

        Console.WriteLine();
        Console.WriteLine("Factorials using nonrecursive method.");
        Console.WriteLine("Factorial of 3 is " + f.FactI(3));
        Console.WriteLine("Factorial of 4 is " + f.FactI(4));
        Console.WriteLine("Factorial of 5 is " + f.FactI(5));
    }
}
```

خروجی:

```
Factorials using recursive method.
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120

Factorials using nonrecursive method.
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120
Press any key to continue . . .
```

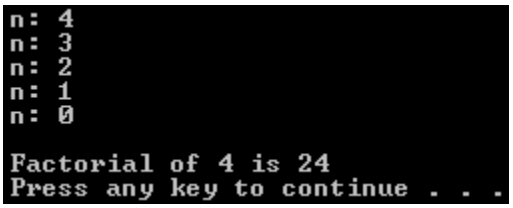
عملکرد متد nonrecursive واضح است، در این متد از یک حلقه استفاده شده که از ۱ شروع شده است و در هر دور متغیر result در t ضرب می‌شود. اما عملکرد متد بازگشتی FactR() اندکی پیچیده‌تر است. هنگامی‌که FactR() با argument ای با مقدار ۱ فراخوانی شود، متد مقدار ۱ را باز می‌گرداند. در غیر این‌صورت حاصل n * FactR(n - 1) را return می‌کند. این عبارت این‌گونه ارزیابی می‌شود که FactR() با مقدار n – 1 فراخوانی شده و این پروسه آن‌قدر تکرار می‌شود که n برابر با ۰ شده و فراخوانی متد منجربه return شود. برای مثال، هنگامی‌که قصد دارید فاکتوریل ۲ را حساب کنید، اولین بار که FactR() اجرا می‌شود argument آن ۲ است که منجربه فراخوانی مجدد FactR() با argument ای با مقدار ۱ و سپس ۰ شده که موجب می‌شود مقدار ۱ return شود. در نهایت این مقدار در ۲ (مقدار اصلی n) ضرب می‌شود و جواب ۲ خواهد بود.

اگر درون متد FactR() یک Console.WriteLine() قرار دهید مقدار n را در هر مرحله خواهید دید:

```
using System;
class Factorial
{
    public int FactR(int n)
    {
        Console.WriteLine("n: " + n);
        if (n == 0) return 1;
        else return FactR(n - 1) * n;
    }
}
class Recursion
```

```
{
    static void Main()
    {
        Factorial f = new Factorial();
        Console.WriteLine("\nFactorial of 4 is " + f.FactR(4));
    }
}
```

خروجی:



در این‌جا، متد با مقدار n = 4 شروع می‌شود. توجه کنید که این متد آن‌قدر تکرار می‌شود تا در نهایت به یک جواب برسد. از آن‌جا که n مخالف صفر است، قسمت else اجرا خواهد شد. ۴ در حافظه نگه داشته شده و متد سعی می‌کند فاکتوریل ۳ را محاسبه کند. فاکتوریل ۳ هنوز جوابی ندارد بنابراین ۳ در حافظه نگه‌داری می‌شود و برنامه سعی می‌کند که فاکتوریل ۲ را به‌دست آورد. ۲ نیز در حافظه ذخیره می‌شود و برنامه سعی می‌کند تا فاکتوریل ۱ را بدست آورد. ۱ در حافظه می‌ماند و ۰ بررسی می‌شود. همان‌طور که می‌بینید اگر به متد مقدار ۰ را بدهید مقدار ۱ return خواهد شد بنابراین این پروسه در نهایت به یک جواب رسیده است و ما اکنون پاسخ فاکتوریل ۰ را داریم که برابر با ۱ است. فاکتوریل ۱ برابر است با !۰ × ۱ و آز آن‌جا که !۰ برابر با ۱ است، !۱ برابر با ۱ خواهد بود. فاکتوریل ۲ برابر است با !۱ × ۲ و از آن‌جا که !۱ برابر با ۱ است، !۲ برابر با ۲ = ۱ × ۲ است. فاکتوریل ۳ نیز برابر است با !۲ × ۳ و از آن‌جا که !۲ برابر با ۲ است، !۳ برابر با ۶ = ۲ × ۳ است. در نهایت فاکتوریل ۴ برابر با ۶ × ۴ خواهد بود.

در مثال بالا چگونه‌گی عمل‌کرد یک متد recursive را مشاهده کردید. همه‌ی متدهای recursive نیز به همین روال عمل می‌کنند. در زیر مثال‌هایی از recursive method می‌بینید که باعث می‌شود درک بهتری نسبت به این موضوع پیدا کنید:

سری فیبوناچی:

```
using System;
class Fibonacci
{
    public long Fib(int n)
    {
        if (n == 0 || n == 1)
            return n;
        return Fib(n - 2) + Fib(n - 1);
    }
}
```

```
class Recursion
{
    static void Main()
    {
        // Fibonacci Sequence
        // 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

        Fibonacci f = new Fibonacci();
        Console.WriteLine(f.Fib(6)); // Output is 8
        Console.WriteLine(f.Fib(7)); // Output is 13
        // and so on...
    }
}
```

نمایش معکوس یک رشته:

```
// Display a string in reverse by using recursion.
using System;
class RevStr
{
    // Display a string backward.
    public void DisplayRev(string str)
    {
        if (str.Length > 0)
            DisplayRev(str.Substring(1, str.Length - 1));
        else
            return;

        Console.Write(str[0]);
    }
}
class RevStrDemo
{
    static void Main()
    {
        string s = "this is a test";
        RevStr rsOb = new RevStr();

        Console.WriteLine("Original string: " + s);

        Console.Write("Reversed string: ");
        rsOb.DisplayRev(s);

        Console.WriteLine();
    }
}
```

خروجی:

```
Original string: this is a test
Reversed string: tset a si siht
Press any key to continue . . .
```

هربار که DisplayRev() فراخوانی می‌شود، ابتدا بررسی می‌شود که آیا str طول بزرگ‌تر از صفر دارد یا خیر. اگر بزرگ‌تر بود، متد DisplayRev() به‌طور recursive با یک رشته‌ی جدید که شامل str به‌جز کاراکتر اول آن است، فراخوانی می‌شود. این پروسه آن‌قدر تکرار می‌شود تا رشته‌ای به طول صفر به متد داده شود. این عمل موجب می‌شود که فراخوانی‌های بازگشتی از ریشه و ابتدا شروع شوند. سپس، اولین کاراکتر str در هر فراخوانی نمایش داده می‌شود. به این ترتیب کل رشته از انتها به ابتدا نمایش داده می‌شوند.

هر مسئله‌ای که به‌طور recursive قابل حل باشد، به احتمال خیلی زیاد از راه nonrecursive نیز قابل حل است. روش recursive در اجرا از روش nonrecursive سرعت کمتری دارد و از طرفی حل بیشتر مسائل از روش recursive ساده و قابل فهم‌تر است. از این‌رو در موقعیت‌هایی که سرعت اجرا به شدت برای شما اهمیت دارد می‌توانید از روش nonrecursive برای حل مسئله استفاده کنید.

نمودار زیر می‌تواند گویای این موضوع باشد: