جلسه سوم- سیشارپ
برنامه نویسی پیشرفته
کاردانی نرم ۱

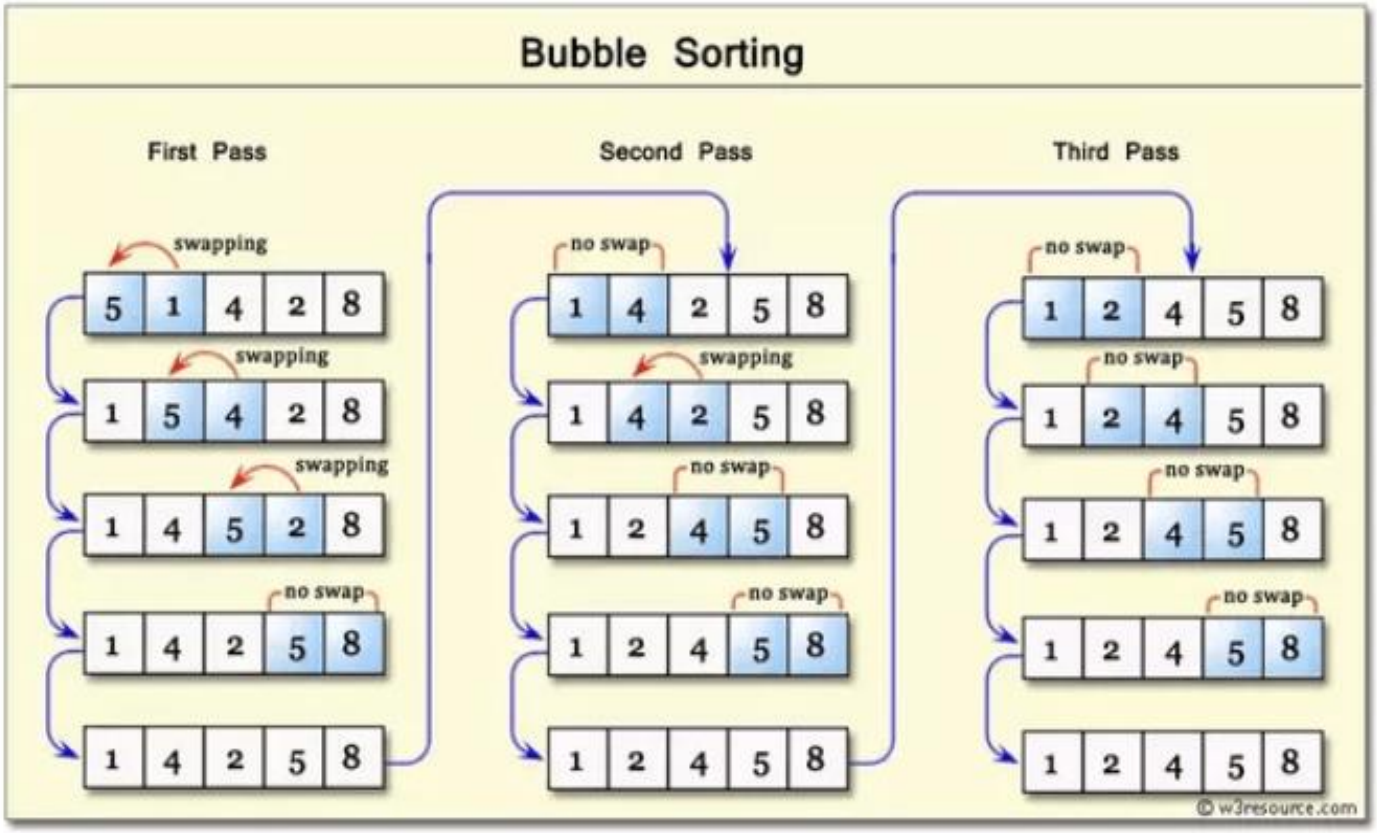
مدرس:

فاطمه دهقانی فیروزآبادی

مرتب سازی حبابی چیست ؟ الگوریتم مرتب سازی حبابی (Bubble Sort)

الگوریتم مرتب‌سازی ساده‌است که لیست را پشت سرهم پیمایش می‌کند تا هر بار عناصر کنارهم را با هم سنجیده و اگر در جای نادرست بودند جابه‌جایشان کند. در این الگوریتم این کار باید تا زمانی که هیچ جابه‌جایی در لیست رخ ندهد، ادامه یابد و در آن زمان لیست مرتب شده‌است. این مرتب‌سازی از آن رو حبابی نامیده می‌شود که هر عنصر با عنصر کناری خود سنجیده‌شده و درصورتی که از آن کوچک‌تر باشد جای خود را به آن می‌دهد و این کار همچنان پیش می‌رود تا کوچک‌ترین عنصر به پایین لیست برسد و دیگران نیز به ترتیب در جای خود قرار گیرند.

مرتب سازی در شکل زیر توضیح داده شده است.



در ادامه کد الگوریتم مرتب سازی حبابی (Bubble Sort) را در سی شارپ برای شما قرار میدهیم.

```
using System;
class bubblesort
{
        static void Main(string[] args)
        {
            int[] a = { 3, 2, 5, 4, 1 };
            int t;
            Console.WriteLine("The Array is : ");
            for (int i = 0; i < a.Length; i++)
            {
                Console.WriteLine(a[i]);
            }
            for (int j = 0; j <= a.Length - 2; j++)
            {
                for (int i = 0; i <= a.Length - 2; i++)
                {
                    if (a[i] > a[i + 1])
                    {
                        t = a[i + 1];
                        a[i + 1] = a[i];
                        a[i] = t;
                    }
                }
            }
            Console.WriteLine("The Sorted Array :");
            foreach (int aray in a)
                Console.Write(aray + " ");
            Console.ReadLine();
        }
    }
```

خروجی کد بالا همانند زیر خواهد بود.

```
The Array is :
3
2
5
4
1
The Sorted Array :
1
2
3
4
5
```

مرتب‌سازی انتخابی چیست ؟ / الگوریتم مرتب سازی انتخابی (Selection)

معمولاً اطلاعات و داده‌های خامی که در اختیار برنامه‌نویس قرار دارد به صورت نامرتب هستند. مواقعی پیش می‌آید که لازم است این داده‌ها بر حسب فیلد خاصی مرتب بشوند؛ مانند لیست دانش آموزان بر حسب معدل، لیست کارمندان بر حسب شماره پرسنلی، لیست دفترچه تلفن بر حسب نام خانوادگی و... روش‌های متعددی برای مرتب‌سازی وجود دارد. برای شروع روش مرتب‌سازی انتخابی (Selection Sort):

روش انتخابی اولین روشی است که به ذهن می‌رسد: بزرگ‌ترین رکورد بین رکوردهای لیست را پیدا می‌کنیم و به انتهای لیست انتقال می‌دهیم. از بقیه رکوردها بزرگ‌ترین را انتخاب می‌کنیم و انتهای لیست - کنار رکورد قبلی - قرار می‌دهیم و... مثلاً:

```
۰:  ۹ ۱ ۶ ۴ ۷ ۳ ۵

۱:  ۵ ۱ ۶ ۴ ۷ ۳ ۹

۲:  ۵ ۱ ۶ ۴ ۳ ۷ ۹

۳:  ۵ ۱ ۳ ۴ ۶ ۷ ۹

۴:  ۴ ۱ ۳ ۵ ۶ ۷ ۹

۵:  ۳ ۱ ۴ ۵ ۶ ۷ ۹

۶:  ۱ ۳ ۴ ۵ ۶ ۷ ۹
```

متد

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آنها استفاده کرد. متدها دارای آرگومانهایی هستند که وظیفه متد را مشخص می‌کنند. متد در داخل کلاس تعریف می‌شود. نمی‌توان یک متد را در داخل متد دیگر تعریف کرد. وقتی که شما در برنامه یک متد را صدا می‌زنید برنامه به قسمت تعریف متد رفته و کدهای آن را اجرا می‌کند. در سی شارپ متدی وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند با ید از کجا شروع شوند، این متد Main() نام دارد.

پارامترها همان چیزهایی هستند که متد منتظر دریافت آنها است.

آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک متد به صورت زیر است :

```
returnType MethodName(Parameter List)
{
   code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک متد برای چاپ یک پیغام در صفحه نمایش استفاده شده است :

```
1  using System;
2
3  public class Program
4  {
5      static void PrintMessage()
6      {
7          Console.WriteLine("Hello World!");
8      }
9
10     public static void Main()
11     {
12         PrintMessage();
13     }
14 }
```

```
Hello World!
```

در خطوط 5-8 یک متد تعریف کرده‌ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می‌توانید آن را زیر متد Main() تعریف کنید. می‌توان این متد را در داخل متد دیگر صدا زد (فراخوانی کرد). متد دیگر ما در اینجا متد Main() است که می‌توانیم در داخل آن نام متدی که برای چاپ یک پیغام تعریف کرده‌ایم (یعنی متد PrintMessage()) را صدا بزنیم. متد Main() به صورت static تعریف شده است. برای اینکه بتوان از متد PrintMessage()در داخل متد Main() استفاده کنیم باید آن را به صورت static تعریف کنیم.

کلمه static به طور ساده به این معناست که می‌توان از متد استفاده کرد بدون اینکه از کلاس نمونه‌ای ساخته شود. متد Main() همواره باید به صورت static تعریف شود چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می‌کند. وقتی به مبحث برنامه نویسی شی گرا رسیدید به طور دقیق کلمه staticمورد بحث قرار می‌گیرد. برنامه class(مثال بالا) زمانی اجرا می‌شود که برنامه دو متدی را که تعریف کرده‌ایم را اجرا کند و متد Main() به صورت static تعریف شود. درباره این کلمه کلیدی در درسهای آینده مطالب بیشتری می‌آموزیم.

در تعریف متد بالا بعد از کلمه static کلمه کلیدی void آمده

است که نشان دهنده آن است که متد مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک متد و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد. نام متد ما PrintMessage() است.

به این نکته توجه کنید که در نامگذاری متد از روش پاسکال (حرف اول هر کلمه بزرگ نوشته می‌شود) استفاده کرده‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید. بهتر است در نامگذاری متدها از کلماتی استفاده شود که کاران متد را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor . همچنین به عنوان مثال اگر مقدار برگشتی متد یک مقدار بولی باشد می‌توانید اسم متد خود را به صورت یک کلمه سوالی انتخاب کنید مانند IsLeapyear یا IsTeenager...ولی از کذاشتن علامت سؤال در آخر اسم متد خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متد است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درسهای آینده در مورد متدها بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متد را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در داخل متد Main() متدی را که در خط 12 ایجاد کرده‌ایم را صدا می‌زنیم. برای صدا زدن یک متد کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم.

اگر متد دارای پارامتر باشد باید شما آرگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درسهای آینده توضیح بیشتری می‌دهیم. با صدا زدن یک متد کدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متد PrintMessage() برنامه از متد Main() به محل تعریف متد PrintMessage() می‌رود. مثلاً وقتی ما متد PrintMessage() را در خط 12 صدا می‌زنیم برنامه از خط 12 به خط 7، یعنی جایی که متد تعریف شده می‌رود. اکنون ما یک متد در برنامه class داریم و همه متدهای این برنامه می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متد

متدها می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متد است و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی متد است. نکته مهم در مورد یک متد، مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متد آسان است. کافیست در تعریف متد به روش زیر عمل کنید :

```
returnType MethodName()
{
   return value;
}
```

returnType در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (int،bool،...). در داخل بدنه متد کلمه کلیدی return و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می‌نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متد و قبل از نام متد ذکر شود. اگر متد ما مقدار برگشتی نداشته باشد باید از کلمه void قبل از نام متد استفاده کنیم. مثال زیر یک متد که دارای مقدار برگشتی است را نشان می‌دهد.

```
1  using System;
2
3  public class Program
4  {
5      static int CalculateSum()
6      {
7          int firstNumber = 10;
8          int secondNumber = 5;
9
10         int sum = firstNumber + secondNumber;
11
12         return sum;
13     }
14
15     public static void Main()
16     {
17         int result = CalculateSum();
18
19         Console.WriteLine("Sum is {0}.", result);
20     }
21 }
```

```
Sum is 15.
```

همانطور که در خط 5 مثال فوق مشاهده می‌کنید هنگام تعریف متد از کلمه int به جای void استفاده کرده‌ایم که نشان دهنده آن است که متد ما دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط 7 و 8 دو متغیر تعریف و مقدار دهی شده‌اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متد Main قابل دسترسی نیستند و فقط در متدی که در آن تعریف شده‌اند قابل استفاده هستند. در خط 10 جمع دو متغیر در متغیر sum قرار می‌گیرد. در خط 12 مقدار برگشتی sum توسط دستور return فراخوانی می‌شود. در داخل متد Main یک متغیربه نام result در خط 17 تعریف می‌کنیم و متد ()CalculateSum را فراخوانی می‌کنیم.

متد ()CalculateSum مقدار 158 را بر می‌گرداند که در داخل متغیر result ذخیره می‌شود. در خط 19 مقدار ذخیره شده در متغیر result چاپ می‌شود.

متدی که در این مثال ذکر شد متد کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در متد بالا نوشته شده ولی همیشه مقدار برگشتی 15 است، در حالیکه می‌توانستیم به راحتی یک متغیر تعریف کرده و مقدار 15 را به آن اختصاص دهیم. این متد در صورتی کارامد است که پارامترهایی به آن اضافه شود که در درسهای آینده توضیح خواهیم داد. هنگامی که می‌خواهیم در داخل یک متد از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه کنید :

```
using System;

public class Program
{
    static int GetNumber()
    {
        int number;

        Console.Write("Enter a number greater than 10: ");
        number = Convert.ToInt32(Console.ReadLine());

        if (number > 10)
        {
            return number;
        }
        else
        {
            return 0;
        }
    }

    public static void Main()
    {
        int result = GetNumber();

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

```
using System;

public class Program
{
    static void TestReturnExit()
    {
        Console.WriteLine("Line 1 inside the method TestReturnExit()");
        Console.WriteLine("Line 2 inside the method TestReturnExit()");

        return;

        //The following lines will not execute
        Console.WriteLine("Line 3 inside the method TestReturnExit()");
        Console.WriteLine("Line 4 inside the method TestReturnExit()");
    }

    public static void Main()
    {
        TestReturnExit();
        Console.WriteLine("Hello World!");
    }
}
```

```
Line 1 inside the method TestReturnExit()
Line 2 inside the method TestReturnExit()
Hello World!
```

در برنامه بالا نحوه خروج از متد با استفاده از کلمه کلیدی return و نادیده گرفتن همه کدهای بعد از این کلمه کلیدی نشان داده شده است. در پایان برنامه متد تعریف شده (TestReturnExit()) در داخل متد Main() فراخوانی و اجرا می‌شود.

پارامترها و آرگومان ها

پارامترها داده‌های خامی هستند که متد آنها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید، در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آنها کارش را به پایان برساند. یک متد می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متد با N پارامتر نشان داده شده است :

```
returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{
   code to execute;
}
```

پارامترها بعد از نام متد و بین پرانتزها قرار می‌گیرند. بر اساس کاری که متد انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متد اضافه کرد. بعد از فراخوانی یک متد باید آرگومانهای آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومانها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومانها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدهید که یک مثال بزنیم :

```
1  using System;
2
3  public class Program
4  {
5      static int CalculateSum(int number1, int number2)
6      {
7          return number1 + number2;
8      }
9
10     public static void Main()
11     {
12         int num1, num2;
13
14         Console.Write("Enter the first number: ");
15         num1 = Convert.ToInt32(Console.ReadLine());
16         Console.Write("Enter the second number: ");
17         num2 = Convert.ToInt32(Console.ReadLine());
18
19         Console.WriteLine("Sum = {0}", CalculateSum(num1, num2));
20     }
21 }
```

```
Enter the first number: 10
Enter the second number: 5
Sum = 15
```

در برنامه بالا یک متد به نام CalculateSum() (خطوط 5-8) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید int باشد. متد دارای دو پارامتر است که اعداد را به آنها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی number1 و number2 مقادیری از نوع اعداد صحیح (int) دریافت می‌کنند. در بدنه متد دستور return نتیجه جمع دو عدد را بر می‌گرداند. در داخل متد Main برنامه از کاربر دو مقدار را درخواست می‌کند و آنها را داخل متغیرها قرار می‌دهد. حال متد را که آرگومانهای آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار num1 به پارامتر اول و مقدار num2 به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار num2 به پارامتر اول و مقدار num1 به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جابه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومانها هنگام فراخوانی متد دقیقاً با ترتیب قرار گیری پارامترها تعریف شده در متد مطابقت داشته باشد.

بعد از ارسال مقادیر 10 و 5 به پارامترها، پارامترها آنها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا camelCasing (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه متد (خط 7) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدی که متد()CalculateSum را فراخوانی می‌کند) ارسال می‌شود. در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط 7). در داخل متد Main از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط 19 متد ()CalculateSum را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آنها برگردانده می‌شود. مقدار برگشت داده شده از متد به وسیله متد ()WriteLine از کلاس Console نمایش داده می‌شود. (خط 19) در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```
1  using System;
2
3  public class Program
4  {
5      static void ShowMessageAndNumber(string message, int number)
6      {
7          Console.WriteLine(message);
8          Console.WriteLine("Number = {0}", number);
9      }
10
11     public static void Main()
12     {
13         ShowMessageAndNumber("Hello World!", 100);
14     }
15 }
```

```
Hello World!
Number = 100
```

در مثال بالا یک متدی تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع int دریافت می‌کند. متد به سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط 13 متد را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متد به صورت زیر فراخوانی می‌شد :

```
ShowMessageAndNumber(100, "Welcome to Gimme C#!");
```

در برنامه خطا به وجود می‌آمد چون عدد 100 به پارامتری از نوع رشته و رشته Hello World! به پارامتری از نوع اعداد صحیح ارسال می‌شد. این نشان می‌دهد که ترتیب ارسال آرگومانها به پارامترها هنگام فراخوانی متد مهم است. به مثال 1 توجه کنید در آن مثال دو عدد از نوع int به پارامترها ارسال کردیم که ترتیب ارسال آنها چون هردو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متد دارای اهداف خاصی باشند ترتیب ارسال آرگومانها مهم است.

```
void ShowPersonStats(int age, int height)
{
    Console.WriteLine("Age = {0}", age);
    Console.WriteLine("Height = {0}", height);
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);
```

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده‌ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید چون سن شخص 20 و قد او 160 سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص 160 و قد او 20 سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارامد تری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

در مثال بالا نشان داده شده است که حتی اگر متد دو آرگومان با یک نوع داده‌ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متد بالا اشکالی به چشم نمی‌آید چون سن شخص 20 و قد او 160 سانتی متر است. اگر آرگومانها را به ترتیب ارسال نکنیم سن شخص 160 و قد او 20 سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومانها باعث می‌شود که شما متدهای کارامد تری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متد را به عنوان آرگومان به متد دیگر ارسال کنید.

```
int MyMethod()
{
   return 5;
}

void AnotherMethod(int number)
{
   Console.WriteLine(number);
}

// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متد MyMethod() عدد 5 است و به عنوان آرگومان به متد AnotherMethod() ارسال می‌شود خروجی کد بالا هم عدد 5 است.

نامیدن آرگومان ها

یکی دیگر از راه‌های ارسال آرگومانها استفاده از نام آنهاست. استفاده از نام آرگومانها شما را از به یاد آوری و رعایت ترتیب پارامترها هنگام ارسال آرگومان‌ها راحت می‌کند. در عوض شما باید نام پارامترهای متد را به خاطر بسپارید (ولی از آن جاییکه ویژوال استودیو Intellisense دارد نیازی به این کار نیست.) استفاده از نام آرگومانها خوانایی برنامه را بالا می‌برد چون شما می‌توانید ببینید که چه مقادیری به چه پارامترهایی اختصاص داده شده است.

نامیدن آرگومانها در سی شارپ 2010 مطرح شده است و اگر شما از نسخه‌های قبلی مانند سی شارپ 2008 استفاده می‌کنید نمی‌توانید از این خاصیت استفاده کنید. در زیر نحوه استفاده از نام آرگومانها وقتی که متد فراخوانی می‌شود نشان داده شده است :

```
MethodToCall( paramName1: value, paramName2: value, ... paramNameN: value);
```

حال به مثال زیر توجه کنید :

```
1  using System;
2
3  public class Program
4  {
5      static void SetSalaries(decimal jack, decimal andy, decimal mark)
6      {
7          Console.WriteLine("Jack's salary is {0:C}.", jack);
8          Console.WriteLine("Andy's salary is {0:C}.", andy);
9          Console.WriteLine("Mark's salary is {0:C}.", mark);
10     }
11
12     public static void Main()
13     {
14         SetSalaries(jack: 120, andy: 30, mark: 75);
15
16         //Print a newline
17         Console.WriteLine();
18
19         SetSalaries(andy: 60, mark: 150, jack: 50);
20
21         Console.WriteLine();
22
23         SetSalaries(mark: 35, jack: 80, andy: 150);
24     }
25 }
```

```
Jack's salary is $120.
Andy's salary is $30.
Mark's salary is $75.

Jack's salary is $50.
Andy's salary is $60.
Mark's salary is $150.

Jack's salary is $80.
Andy's salary is $150.
Mark's salary is $35.
```

متد ()WriteLine در خطوط 7-9 از فرمت پول رایج که با {0:C} نشان داده می‌شود استفاده کرده است که یک داده عددی را به نوع پولی تبدیل می‌کند. خروجی نشان می‌دهد که حتی اگر ما ترتیب آرگومانها در سه متد فراخوانی شده را تغییر دهیم مقادیر مناسب به پارامترهای مربوطه‌شان اختصاص داده می‌شود. همچنین می‌توان از آرگومانهای دارای نام و آرگومانهای ثابت (مقداری) به طور همزمان استفاده کرد به شرطی که آرگومانهای ثابت قبل از آرگومانهای دارای نام قرار بگیرند.

```
//Assign 30 for Jack's salary and use named arguments for
// the assignment of the other two

SetSalary(30, andy: 50, mark: 60);

// or

SetSalary(30, mark: 60, andy: 50);

//The following codes are wrong and will lead to errors

SetSalary(mark: 60, andy: 50, 30);

// and

SetSalary(mark: 60, 30, andy: 50);
```

```
//Assign 30 for Jack's salary and use named arguments for
// the assignment of the other two

SetSalary(30, andy: 50, mark: 60);

// or

SetSalary(30, mark: 60, andy: 50);

//The following codes are wrong and will lead to errors

SetSalary(mark: 60, andy: 50, 30);

// and

SetSalary(mark: 60, 30, andy: 50);
```

همانطور که مشاهده می‌کنید ابتدا باید آرگومانهای ثابت هنگام فراخوانی متد ذکر شوند. در اولین و دومین فراخوانی در کد بالا، مقدار 30 را به عنوان اولین آرگومان به اولین پارامتر متد یعنی jack اختصاص می‌دهیم. سومین و چهارمین خط کد بالا اشتباه هستند چون آرگومانهای دارای نام قبل از آرگومانهای ثابت قرار گرفته‌اند. قرار گرفتن آرگومانهای دارای نام بعد از آرگومانها ثابت از بروز خطا جلوگیری می‌کند.

تمرین

- یک تابع تعریف کنید که دو عدد می گیرد و کوچکترین آنها را برمی گرداند