

# Phương thức thành viên (member method) của class C#

[Hướng dẫn tự học lập trình C# toàn tập](#) > **Phương thức thành viên (member method) của class C#**

Phương thức trong C# – thành phần xử lý thông tin của struct hoặc class – đã được xem xét một phần trong bài học về [struct trong C#](#). Tuy nhiên, còn rất nhiều vấn đề quan trọng cần biết khi xây dựng và sử dụng phương thức trong C#. Bài học này sẽ tiếp tục cung cấp những thông tin chi tiết hơn về phương thức của C#, tập trung vào một vấn đề quan trọng: tham số của phương thức.

## NỘI DUNG CỦA BÀI [ Ấn ]

1. Phương thức thành viên trong C#
  - 1.1. Khai báo phương thức
  - 1.2. Kiểu dữ liệu trả về
  - 1.3. Danh sách tham số
  - 1.4. Gọi phương thức trong C#
2. Tham số của phương thức trong C#: value type và reference type
  - 2.1. Truyền kiểu giá trị
  - 2.2. Truyền kiểu tham chiếu, từ khóa ref
  - 2.3. Ví dụ về truyền tham số cho phương thức
3. Tham số out
  - 3.1. Tham số out là gì?
  - 3.2. Khai báo phương thức với tham số out
4. Tham số tùy chọn
5. Tham số params
6. Nạp chồng phương thức (method overloading) trong C#
  - 6.1. Nạp chồng phương thức trong C# là gì?
  - 6.2. Signature của phương thức trong C#
7. Một số vấn đề khác của phương thức trong C#
  - 7.1. Phương thức với Expression body
  - 7.2. Named Arguments
8. Kết luận

## Phương thức thành viên trong C#

Như bạn đã biết, *phương thức* (method) trong C# là một thành viên của class (và struct), là một khối code được đặt tên và chứa các lệnh để cùng thực hiện một nhiệm vụ cụ thể. Phương thức cho phép tái sử dụng code mà không phải viết lặp đi lặp lại nhiều lần.

Phương thức trong C# tương tự như hàm (function) và thủ tục (procedure) của Pascal, chương trình con Sub của visual basic, v.v.. Sự khác biệt lớn là phương thức của C# bắt buộc phải là thành viên của một kiểu dữ liệu như struct hay class. Trong C# không có phương thức “tự do” hay “toàn cục”.

Làm việc với phương thức chia làm hai giai đoạn:

- Khai báo (định nghĩa): Ở *giai đoạn khai báo* chúng ta mô tả các thông tin bắt buộc về phương thức, cũng như viết các lệnh cần thực hiện trong thân phương thức.
- Gọi (sử dụng): đây là giai đoạn chúng ta cung cấp dữ liệu thực sự để phương thức thực hiện những lệnh đã được thiết kế sẵn ở phần định nghĩa.

## Khai báo phương thức

Cấu trúc chung để khai báo phương thức trong class C# như sau:

```
[public|protected|private] <kiểu ra> <tên phương thức> ([danh sách tham số])  
{  
    [thân phương thức]  
    [return [giá trị];]  
}
```

Trong đó:

`public`, `protected` và `private` là các từ khóa điều khiển truy cập tương tự như đối với biến thành viên. Mặc định C# xem phương thức là `private` nếu không có từ khóa nào được chỉ định.

So với struct, phương thức thành viên class có thêm từ khóa `protected`.

*Tên phương thức* do người dùng tự đặt và tuân thủ theo quy tắc đặt định danh của C#. Ngoài ra, trong C# quy ước tên phương thức viết theo kiểu PascalCase (luôn bắt đầu bằng chữ in hoa).

## Kiểu dữ liệu trả về

*Kiểu trả về* là kiểu dữ liệu của kết quả nhận được sau khi kết thúc thực hiện các lệnh trong thân phương thức. Kiểu trả về có thể là bất kỳ kiểu dữ liệu nào được C#/.NET định nghĩa sẵn hoặc cũng có thể là những kiểu dữ liệu do người dùng định nghĩa.

Cũng có thể có trường hợp phương thức không trả về kết quả nào (ví dụ, chúng ta chỉ yêu cầu phương thức viết thông tin ra màn hình). Khi đó, C# yêu cầu phải viết kiểu trả về là `void`, là một từ khóa của C#.

Nếu kiểu trả về khác `void`, trong thân phương thức bắt buộc phải có lệnh `return` để báo rằng, giá trị đi sau `return` sẽ là kết quả thực hiện của phương thức. Nếu kiểu trả về là `void` thì không bắt buộc phải có `return`.

## Danh sách tham số

*Danh sách tham số* (còn gọi là danh sách tham số hình thức) là danh sách biến có thể sử dụng trong phương thức.

Ở giai đoạn định nghĩa phương thức, chúng ta không biết giá trị cụ thể của các biến này mà chỉ có thể sử dụng tên biến trong các lệnh ở thân phương thức.

Ở giai đoạn gọi phương thức người sử dụng phương thức mới cung cấp các giá trị cụ thể (gọi là tham số thực).

Vì lý do này, ở giai đoạn định nghĩa thường phải kiểm tra hết các tình huống có thể xảy ra với tham số hình thức.

Danh sách tham số được định nghĩa theo quy tắc sau:

```
(<kiểu_1> <tham_số_1>, <kiểu_2> <tham_số_2>, ...)
```

Hình dung một cách đơn giản, danh sách tham số chính là một chuỗi khai báo biến cục bộ viết tách nhau bởi dấu phẩy. Do đó, mỗi tham số đều tuân thủ quy tắc khai báo:

```
<kiểu_dữ_liệu> <tên_biến>
```

Danh sách tham số không bắt buộc phải có trong khai báo phương thức. Nếu danh sách tham số trống, ta chỉ cần viết cặp dấu ngoặc tròn sau tên phương thức.

## Gọi phương thức trong C#

Phương thức trong C# chỉ có thể khai báo là thành viên của một class hoặc struct nào đó (trừ phương thức lambda, phương thức vô danh và hàm cục bộ sẽ xem xét sau). Khai báo phương thức trong C# không thể nằm trực tiếp trong namespace, không thể nằm ngoài namespace, cũng không thể nằm ngoài class/struct.

Ở *giai đoạn sử dụng* (gọi phương thức), người dùng cung cấp giá trị đầu vào thực (nếu có) và nhận giá trị trả về (nếu có) qua "lời gọi phương thức" theo cấu trúc:

```
<tên_phương_thức>([biến_1, biến_2, ...]);
```

Trong đó biến 1, biến 2, v.v. phải có kiểu theo đúng trật tự như khi định nghĩa phương thức. Danh sách biến cung cấp cho lời gọi phương thức gọi là các tham số thực (để phân biệt với danh sách tham số hình thức khi khai báo phương thức).

Trong phần tiếp theo chúng ta sẽ xem xét thêm một số vấn đề khác của phương thức (như truyền tham biến/tham trị, tham số ra, danh sách tham số biến đổi, giá trị tham số mặc định, v.v.).

## Tham số của phương thức trong C#: value type và reference type

Như bạn đã biết, các kiểu dữ liệu của C# chia làm hai loại: value type và reference type. Object của value type nằm trong stack, còn object của reference type nằm trong heap. Hai loại kiểu này biểu hiện khác nhau khi sử dụng trong tham số của phương thức.

## Truyền kiểu giá trị

Khi truyền một biến thuộc kiểu giá trị cho một phương thức, một bản sao của biến này được tạo ra trong stack của phương thức được gọi. Tất cả các thao tác mà phương thức thực hiện trên tham số này đều chỉ tác động trên bản sao.

Do đó, sau khi kết thúc phương thức, giá trị của biến tham số vẫn giữ nguyên như trước khi truyền vào phương thức. Tức là những thay đổi (nếu có) của biến trong phương thức không được giữ lại.

Ví dụ, nếu truyền giá trị `i = 100` cho phương thức, một bản sao của `i` được tạo ra và truyền vào phương thức. Những gì thay đổi trong thức thực chất đều tác động lên bản sao của `i`, chứ không phải chính `i`. Do vậy, khi kết thúc phương thức, bản sao bị hủy bỏ, còn `i` không thay đổi gì.

## Truyền kiểu tham chiếu, từ khóa `ref`

Khi truyền một biến thuộc kiểu tham chiếu, bản thân địa chỉ của vùng heap nơi lưu giá trị đó được truyền vào cho phương thức. Tất cả những thao tác trên biến tham số đó thực chất đều tác động thẳng lên giá trị nằm trong heap. Vì vậy, sau khi kết thúc phương thức, những thay đổi này vẫn được lưu lại.

Từ khóa `ref` cho phép truyền một biến thuộc kiểu giá trị nhưng có thể lưu giữ thay đổi như khi sử dụng biến thuộc kiểu tham chiếu. Khi khai báo phương thức, nếu tham số truyền vào thuộc kiểu giá trị nhưng cần phải giữ lại những thay đổi thực hiện trong thân phương thức, C# cho phép sử dụng từ khóa `ref` trước khai báo tham số đó.

## Ví dụ về truyền tham số cho phương thức

Hãy cùng thực hiện ví dụ sau và đọc kỹ comment để hiểu rõ sự khác biệt giữa tham số kiểu giá trị và kiểu tham chiếu, cũng như tác dụng của từ khóa `ref`.

Tạo một blank solution `S07_Methods` và thêm vào project `P01_Parameters`. Viết code cho `Program.cs` như sau:

```
1. using System;
2. namespace P01_Parameters
3. {
4.     internal class Data
5.     {
6.         public int Id { get; set; }
7.         public string Name { get; set; }
8.     }
9.
10.    internal class ParameterPassingTest
11.    {
12.        // truyền tham số kiểu value
13.        public void MethodWithValueType(int a)
14.        {
15.            a += 10; // thay đổi giá trị tham số
16.        }
17.        // truyền tham số kiểu reference
18.        public void MethodWithReferenceType(Data s)
19.        {
20.            // thay đổi giá trị tham số
21.            s.Name += " Edited";
22.            s.Id += 10;
23.        }
24.    }
25. }
```

```

24.     public void Method2WithReferenceType(Data s)
25.     {
26.         // khởi tạo object khác cho s,
27.         // tương đương với việc cho s tham chiếu sang vùng nhớ khác
28.         s = new Data { Id = 2, Name = "Donald Trump" };
29.     }
30.     // sử dụng từ khóa ref cho kiểu value
31.     public void Method1WithRefKeyword(ref int a)
32.     {
33.         a += 10;
34.     }
35.     // sử dụng từ khóa ref cho kiểu reference
36.     public void Method2WithRefKeyword(ref Data s1, ref Data s2)
37.     {
38.         // chỉ đổi giá trị
39.         s1.Id += 10; s1.Name += " Edited";
40.         // đổi thành một object khác (thay đổi địa chỉ tham chiếu tới)
41.         s2 = new Data { Id = 100, Name = "Donald Trump" };
42.     }
43. }
44.
45. internal class Program
46. {
47.     private static void Main()
48.     {
49.         ParameterPassingTest test = new ParameterPassingTest();
50.         int a = 0;
51.         test.MethodWithValueType(a);
52.         Console.WriteLine(a); // a = 0, không thay đổi, vì a là biến kiểu value
53.         Data d = new Data { Id = 0, Name = "Hello world" };
54.         test.MethodWithReferenceType(d);
55.         // Id = 10, Name = Hello world Edited, giá trị thay đổi vì d thuộc kiểu ref
56.         Console.WriteLine($"{d.Id}, {d.Name}");
57.         // phương thức này lại không làm thay đổi d
58.         // d vẫn giữ giá trị Id = 10, Name = Hello world Edited
59.         test.Method2WithReferenceType(d);
60.         Console.WriteLine($"{d.Id}, {d.Name}"); // Id = 10, Name = Hello world Edited
61.         // như vậy, địa chỉ d trở tới không đổi
62.         test.Method1WithRefKeyword(ref a);
63.         Console.WriteLine(a); // a = 10, đã thay đổi, vì từ khóa ref
64.         Data d2 = new Data { Id = 1, Name = "Barrack Obama" };
65.         test.Method2WithRefKeyword(ref d, ref d2);
66.         // d chỉ thay đổi giá trị, giống trường hợp trên
67.         Console.WriteLine($"{d.Id}, {d.Name}"); // Id = 10, Name = Hello world Edited
68.         // d2 trỏ sang object khác
69.         Console.WriteLine($"{d2.Id}, {d2.Name}"); // Id = 100, Name = Donald Trump
70.         // như vậy, từ khóa ref cho phép kiểu tham chiếu thay đổi cả địa chỉ vùng nhớ
71.         Console.ReadKey();
72.     }
73. }
74. }

```

Qua ví dụ trên chúng ta thấy từ khóa `ref` giúp chúng ta truyền tham số thuộc kiểu value nhưng lại có thể giữ lại những thay đổi đã thực hiện trong phương thức.

Đối với kiểu reference, nếu trong thân phương thức chỉ thay đổi giá trị các thành viên của biến tham số thì những thay đổi này sẽ được lưu lại sau khi kết thúc phương thức.

Tuy nhiên, nếu trong thân phương thức chúng ta thay đổi địa chỉ biến đó trỏ tới (ví dụ, bằng lệnh khởi tạo object mới) thì sự thay đổi này lại không được lưu giữ. Lý do là vì địa chỉ của một vùng nhớ cũng có thể xem là một dạng biến value (thực chất địa chỉ thuộc kiểu số nguyên), do đó không thể thay đổi.

Khi sử dụng từ khóa `ref` với kiểu reference, chúng ta có thể đổi cả địa chỉ mà biến đó trỏ tới.

Qua ví dụ trên chúng ta cũng lưu ý, nếu trong khai báo phương thức sử dụng từ khóa `ref` trước tham số nào thì khi gọi phương thức cũng phải sử dụng từ khóa `ref` trước tham số

tương ứng.

## Tham số out

### Tham số out là gì?

Qua các phần trên bạn có thể thấy, mỗi phương thức có thể nhận một danh sách biến hình thức cung cấp **thông tin đầu vào** cho phương thức. Các biến này sẽ nhận giá trị thực khi gọi phương thức. Đây là cách sử dụng mặc định của tham số. Tham số này cũng được gọi là **tham số vào**.

Kết quả thực hiện của phương thức được trả về thông qua lời gọi phương thức. Bình thường phương thức chỉ có thể trả về một giá trị thông qua lời gọi phương thức.

Có trường hợp ta muốn nhận nhiều hơn một giá trị từ việc thực hiện phương thức. Hãy cùng thử một phương thức đặc biệt: TryParse. TryParse là một phương thức gặp trong hầu hết các struct cơ bản như int, bool. Nó có nhiệm vụ biến đổi chuỗi về kiểu dữ liệu tương ứng.

```
C# Interactive
1. > string input = "12345";
2. > if(int.TryParse(input, out int i))
3. . {
4. .     Console.WriteLine("Success!");
5. .     Console.WriteLine(i++);
6. . }
7. Success!
8. 12345
```

Phương thức TryParse sẽ thử chuyển đổi chuỗi input sang kiểu đích. Nếu chuỗi hợp lệ và chuyển đổi thành công phương thức sẽ trả về giá trị true; nếu bị lỗi, phương thức sẽ trả về giá trị false. Giá trị số nguyên kết quả của việc biến đổi này sẽ được gán cho biến i kiểu int. Như vậy, tham số thứ hai của TryParse giờ không cung cấp dữ liệu đầu vào, mà trở thành nơi chứa dữ liệu đầu ra của phương thức.

Việc gọi phương thức TryParse như vậy giúp người lập trình kiểm tra được kết quả thực hiện mà không bị lỗi dừng chương trình. Nó hoạt động tốt hơn nhiều so với phương thức Parse với cùng chức năng.

C# cung cấp một tính năng đặc biệt gọi là *tham số ra* (out parameter): nếu trước một tham số trong định nghĩa phương thức đặt từ khóa *out*, tham số đó có thể giữ lại giá trị nó có được trong quá trình thực hiện phương thức, và qua đó có thể dùng để chứa kết quả thực hiện của các lệnh trong thân phương thức.

### Khai báo phương thức với tham số out

Cùng xem xét ví dụ sau đây để hiểu rõ hơn về cách định nghĩa và sử dụng của tham số out:

```
1. using System;
2. namespace P02_OutParam
3. {
4.     internal class Program
5.     {
6.         /// <summary>
```

```

7.    /// Thực hiện 3 phép toán trong cùng một phương thức
8.    /// </summary>
9.    /// <param name="a"></param>
10.   /// <param name="b"></param>
11.   /// <param name="sum">tổng (tham số ra)</param>
12.   /// <param name="product">tích (tham số ra)</param>
13.   /// <param name="div">thương (tham số ra)</param>
14.   /// <returns>>true nếu b != 0, false nếu b == 0 (không thực hiện được phép chia)<
15.   private static bool DoMath(int a, int b, out int sum, out int product, out float
16.   {
17.       sum = a + b;
18.       product = a * b;
19.       if (b == 0)
20.       {
21.           div = float.NaN;
22.           return false;
23.       }
24.       div = a / b;
25.       return true;
26.   }
27.   private static void Main(string[] args)
28.   {
29.       int sum, product;
30.       float div;
31.       // người dùng nhập a, b từ bàn phím và biến đổi kiểu thành int
32.       int a = int.Parse(Console.ReadLine());
33.       int b = int.Parse(Console.ReadLine());
34.       // gọi phương thức DoMath,
35.       // kết quả hiển thị phụ thuộc giá trị thu được khi gọi phương thức
36.       bool result = DoMath(a, b, out sum, out product, out div);
37.       Console.WriteLine($"Sum = {sum}");
38.       Console.WriteLine($"Product = {product}");
39.       if (result == true)
40.       {
41.           // nếu phép chia không có lỗi thì in kết quả
42.           Console.WriteLine($"Division = {div}");
43.       }
44.       else
45.       {
46.           // nếu phép chia có lỗi thì báo "chia cho 0"
47.           Console.WriteLine("Division by zero!!!!!!");
48.       }
49.       Console.ReadKey();
50.   }
51. }
52. }

```

Khi sử dụng tham số ra có một số vấn đề sau cần lưu ý:

1. Tham số nào được xác định là tham số ra thì trước khi gọi phương thức phải khai báo biến tương ứng. Biến này sẽ được truyền vào cho phương thức và sẽ lưu lại kết quả sau khi phương thức thực hiện xong.
2. Phải dùng từ khóa out cho cả giai đoạn định nghĩa và giai đoạn gọi phương thức.
3. Tham số ra bắt buộc phải được gán giá trị trong thân phương thức.
4. Biến được khai báo là tham số ra sẽ không bắt buộc phải gán giá trị trước.

C# 7 cho phép khai báo và truyền tham số out trực tiếp trong lời gọi phương thức. Trong ví dụ trên, từ C# 7 bạn có thể thực hiện lời gọi sau đây:

```
bool result = DoMath(a, b, out int sum, out int product, out float div);
```

Lời gọi phương thức này sẽ thực hiện khai báo luôn biến sum, product và div. Sau khi kết thúc phương thức DoMath, các biến này sẽ nhận được giá trị từ thân phương thức. Bạn không cần khai báo riêng rẽ các biến này trước khi gọi phương thức. Thay đổi này giúp việc gọi các phương thức có tham số out đơn giản hơn rất nhiều.

# Tham số tùy chọn

Trước hết hãy cùng thực hiện một ví dụ.

```
1.  using System;
2.
3.  namespace P04_OptionalParameter
4.  {
5.      internal class ConsoleHelper
6.      {
7.          /// <summary>
8.          /// Xuất thông tin ra console với màu sắc (WriteLine có màu)
9.          /// </summary>
10.         /// <param name="message"></param>
11.         /// <param name="bgColor"></param>
12.         /// <param name="fgColor"></param>
13.         /// <param name="resetColor"></param>
14.         public void WriteLine(object message, ConsoleColor bgColor = ConsoleColor.Black,
15.         {
16.             Console.ForegroundColor = fgColor;
17.             Console.BackgroundColor = bgColor;
18.             Console.WriteLine(message);
19.             if (resetColor)
20.                 Console.ResetColor();
21.         }
22.     }
23.
24.     internal class Program
25.     {
26.         private static void Main(string[] args)
27.         {
28.             Console.Title = "Optional parameters";
29.             var helper = new ConsoleHelper();
30.             helper.WriteLine("Hello world from C#");
31.             helper.WriteLine("Hello world from C#", ConsoleColor.Cyan);
32.             helper.WriteLine("Hello world from C#", ConsoleColor.Cyan, ConsoleColor.Magenta);
33.
34.             Console.ReadKey();
35.         }
36.     }
37. }
```

Trong ví dụ trên chúng ta xây dựng phương thức WriteLine với danh sách tham số có chút khác biệt với những phương thức bình thường. Trong hai phương thức này, tham số `bgColor`, `fgColor` và `resetColor` được gán sẵn giá trị: `bgColor = ConsoleColor.Black`, `fgColor = White` và `resetColor = true`.

Tính năng này của C# được gọi là *tham số với giá trị mặc định* hoặc *tham số không bắt buộc* hoặc *tham số tùy chọn* (Optional Arguments / Parameters).

*Tham số tùy chọn* là loại tham số đã được gán sẵn giá trị mặc định khi định nghĩa phương thức, và do đó khi gọi phương thức có thể bỏ qua việc truyền tham số này.

Tham số tùy chọn không có gì khác biệt với tham số bình thường khi sử dụng trong thân phương thức. Tuy nhiên, C# bắt buộc các tham số tùy chọn phải nằm cuối cùng trong danh sách tham số.

Khi gọi phương thức, nếu không cần truyền giá trị khác với giá trị mặc định, có thể bỏ qua tham số tùy chọn này.


Các phương thức Write và WriteLine ở trên mặc dù được định nghĩa với 3 tham số vào nhưng hai tham số sau là tham số tùy chọn: một tham số nhận màu sắc mặc định là White;



một tham số nhận giá trị mặc định là true.

Do đó, khi gọi các phương thức này trong `Main`, chúng ta chỉ cung cấp giá trị cho tham số `color` (do chúng ta muốn viết ra chữ màu Magenta, khác với màu White mặc định) nhưng không cung cấp giá trị cho tham số thứ 3 (vì vẫn muốn dùng giá trị true, vốn là giá trị có sẵn của tham số tùy chọn này).

```
1 reference
private void WriteLine(string message, ConsoleColor color = ConsoleColor.White, bool resetColor = true)
{
    Console.ForegroundColor = color;
    Console.WriteLine(message);
    if (resetColor)
        Console.ResetColor();
}
```

 void BookCreateView.WriteLine(string message, [ConsoleColor color = ConsoleColor.White], [bool resetColor = true])  
xuất thông tin ra console với màu sắc (WriteLine có màu)

Cách Visual Studio hiển thị thông tin hỗ trợ của tham số tùy chọn.

## Tham số params

Bình thường, danh sách tham số của phương thức là cố định. Nó có nghĩa là, nếu bạn khai báo phương thức với, giả sử, 3 tham số, khi gọi phương thức, bạn phải cung cấp đúng 3 tham số theo đúng thứ tự về kiểu.

Giờ hãy nghĩ một tình huống khác. Bạn cần viết một phương thức để cộng các số. Nếu bạn cần cộng một số lượng không giới hạn số thì phải làm sao? Rõ ràng, cách thức sử dụng danh sách tham số bình thường không làm được việc này.

C# cung cấp khả năng viết phương thức mà có thể tiếp nhận số lượng không hạn chế tham số. Hãy cùng xem một ví dụ.

Tạo project mới trong solution và đặt tên là P03\_Params. Viết code cho Program.cs như sau:

```
1. using static System.Console;
2.
3. namespace P03_Params
4. {
5.     internal class Math
6.     {
7.         public double Sum(params double[] operands)
8.         {
9.             var sum = 0.0;
10.            foreach (var o in operands)
11.            {
12.                sum += o;
13.            }
14.            return sum;
15.        }
16.
17.        public double Product(params double[] operands)
18.        {
19.            var product = 1.0;
20.            foreach (var o in operands)
21.            {
22.                product *= o;
23.            }
24.            return product;
25.        }
26.    }
27.
28.    class Message
29.    {
30.        public string Greeting(params string[] message)
```

```

31.     {
32.         var greeting = string.Join(" ", message);
33.         return greeting;
34.     }
35. }
36.
37. internal class Program
38. {
39.     private static void Main(string[] args)
40.     {
41.         Title = "params";
42.
43.         var math = new Math();
44.         var sum1 = math.Sum(1, 2, 3, 4, 5, 6);
45.         var sum2 = math.Sum(1, 2, 3);
46.         var product1 = math.Product(7, 8, 9, 10);
47.         var product2 = math.Product(4, 5, 6);
48.         WriteLine($"Sum(1, 2, 3, 4, 5) = {sum1}");
49.         WriteLine($"Sum(1, 2, 3) = {sum2}");
50.         WriteLine($"Product(4, 5, 6) = {product2}");
51.
52.         var msg = new Message();
53.         var greeting1 = msg.Greeting("Hello", "world", "from", "C#");
54.         var greeting2 = msg.Greeting("Hi", "this", "is", "params", "method");
55.         WriteLine(greeting1);
56.         WriteLine(greeting2);
57.
58.         ReadKey();
59.     }
60. }
61. }

```

Trong ví dụ này, bạn đã tạo một class Math với hai phương thức Sum và Product, class Message với phương thức Greeting.

Điều đặc biệt của các phương thức này nằm ở danh sách tham số. Tất cả chúng đều có cùng cú pháp:

```
(params <type>[] <name>)
```

Cụ thể là

```

double Sum(params double[] operands)
double Product(params double[] operands)
string Greeting(params string[] messages)

```

Đây là cách khai báo của loại tham số đặc biệt: tham số params. Cách khai báo tham số này cho phép bạn cung cấp số lượng không hạn chế tham số (thực) cùng loại khi gọi phương thức:

```

var sum1 = math.Sum(1, 2, 3, 4, 5, 6);
var sum2 = math.Sum(1, 2, 3);
var product1 = math.Product(7, 8, 9, 10);
var product2 = math.Product(4, 5, 6);
var greeting1 = msg.Greeting("Hello", "world", "from", "C#");
var greeting2 = msg.Greeting("Hi", "this", "is", "params", "method");

```

Trong thân phương thức, bạn có thể sử dụng loại tham số này như một mảng một chiều.

```

foreach (var o in operands)
{

```

```
sum += 0;  
}
```

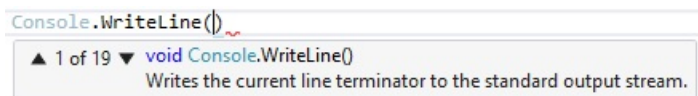
## Nạp chồng phương thức (method overloading) trong C#

### Nạp chồng phương thức trong C# là gì?

*Nạp chồng phương thức* (method overloading) là hiện tượng trong một class có thể tồn tại nhiều phương thức trùng tên.

Nạp chồng phương thức cùng với [nạp chồng toán tử](#) (operator overloading) thuộc về nguyên lý *đa hình tĩnh* (static polymorphism).

Ví dụ phương thức WriteLine của lớp Console có 19 overload khác nhau:



19 overload của phương thức WriteLine

Mỗi overload này nhận một danh sách tham số khác nhau. Intellisense của Visual studio hiển thị tất cả các overload của một phương thức vào một danh sách như trên. Bạn có thể dùng phím mũi tên Up và Down để duyệt qua danh sách này. Ứng với mỗi overload sẽ cung cấp thông tin chi tiết riêng.

Khi gặp hiện tượng nạp chồng phương thức, trình biên dịch của C# sẽ căn cứ vào danh sách tham số thực của lời gọi hàm để quyết định xem người lập trình đang muốn gọi phương thức nào.

Vì vậy, các phương thức nạp chồng bắt buộc phải khác nhau về danh sách tham số. Nói một cách chính xác hơn, các phương thức nạp chồng trong C# bắt buộc phải khác nhau về **signature**. Ngược lại, C# compiler sẽ báo lỗi định nghĩa phương thức trùng nhau.

Tiếp theo đây bạn sẽ biết signature của phương thức là gì.

### Signature của phương thức trong C#

Signature của phương thức trong C# bao gồm các thông tin sau:

- Tên của phương thức;
- Số lượng tham số;
- Kiểu và trật tự của các tham số;
- Các từ khóa điều khiển cho tham số (out, ref, in).

Kiểu dữ liệu trả về không thuộc về signature của phương thức. Tên của tham số hình thức cũng không được tính vào signature của phương thức. Rất nhiều bạn nhầm lẫn hai vấn đề này.

Not part of signature  
↓  
`long AddValues( int a, out int b) { ... }`  
↑  
Signature

C# bắt buộc trong cùng một class không được phép có hai phương thức trùng nhau về signature.

Trong hiện tượng nạp chồng, tên của phương thức trùng nhau, do đó ít nhất 1 trong 3 yếu tố còn lại phải khác nhau. Cả ba yếu tố này đều liên quan đến danh sách tham số. Nói cách khác, các phương thức nạp chồng phải có danh sách tham số khác nhau.

Ví dụ: (string s, int i, bool b) và (string str, int ii, bool bb) là hai danh sách tham số giống nhau:

1. Cả hai đều có 3 tham số;
2. Thứ tự tham số tính theo kiểu đều là (string, int, bool);
3. Tên các tham số không quan trọng.

Nói tóm lại, bạn được phép khai báo các phương thức nạp chồng (trùng tên) nhưng 3 yếu tố còn lại của signature phải khác nhau. Điều này có nghĩa là các phương thức nạp chồng (trùng tên) phải thỏa mãn ít nhất một trong số các điều kiện:

- Số lượng tham số khác nhau;
- Thứ tự tham số tính theo kiểu (không phải tính theo tên) khác nhau;
- Sử dụng modifier khác nhau.

```
class B
{
    long AddValues( long a, long b) { return a+b; }
    int AddValues( long c, long d) { return c+d; } // Error, same signature
}
```

Signature

## Một số vấn đề khác của phương thức trong C#

### Phương thức với Expression body

Nếu thân phương thức chỉ có một lệnh duy nhất, C# cho phép viết phương thức đó ở dạng đơn giản hóa, gọi là **expression body**. Cách viết expression body loại bỏ cặp dấu {} và

lệnh return (nếu có) ở thân phương thức. Expression body sử dụng toán tử `=>` để ghép tên và thân phương thức.

Hãy cùng xem ví dụ:

```
public bool IsSquare(Rectangle rect) => rect.Height == rect.Width;
```

Đây là cách viết ở dạng expression body. Cách viết “truyền thống” của phương thức trên là:

```
public bool IsSquare(Rectangle rect)
{
    return rect.Height == rect.Width;
}
```

Như vậy, nếu thân của phương thức (thông thường) chứa đúng 1 lệnh return, bạn chỉ cần viết biểu thức tính giá trị đó và ghép với tên phương thức qua dấu `=>`. C# sẽ tự hiểu cần trả lại giá trị của biểu thức cho lời gọi phương thức.

Nếu thân phương thức là một lệnh không trả về giá trị (kiểu trả về là void), bạn cũng chỉ cần viết đúng lệnh đó và ghép với tên phương thức bằng dấu `=>`.

## Named Arguments

Ở trên bạn đã biết cách gọi phương thức bằng cách cung cấp danh sách giá trị theo đúng thứ tự về kiểu:

```
// khai báo
public void MoveAndResize(int x, int y, int width, int height) { ... }

// gọi phương thức
r.MoveAndResize(30, 40, 20, 40);
```

C# cho phép gọi phương thức theo một cách khác:

```
r.MoveAndResize(x: 30, y: 40, width: 20, height: 40);
```

Cách gọi phương thức này có tên là **named arguments**. Trong cách gọi này, mỗi tham số được truyền bằng cách viết tên tham số (đặt khi khai báo phương thức), dấu hai chấm và giá trị. Cách gọi này không cần quan tâm về thứ tự viết tham số.

Cách gọi này rất hữu ích nếu phương thức có nhiều tham số hoặc khi sử dụng kết hợp với tham số tùy chọn.

## Kết luận

Bài học này đã cung cấp cho bạn đầy đủ kiến thức cần thiết để làm việc với phương thức trong C#. Những kỹ thuật này có thể sử dụng để xây dựng cả class và struct. Đây cũng là những kỹ thuật hết sức quan trọng để bạn có thể xây dựng class hữu dụng.

- + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
  - + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
  - + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
- Cảm ơn bạn!