Bài: Cấu trúc của hàm cơ bản trong C#

Xem bài học trên website để ủng hộ Kteam: Cấu trúc của hàm cơ bản trong C#

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về các CẤU TRÚC VÒNG LẶP TRONG C#. Trong quá trình viết code, chúng ta thường thấy code của mình quá dài, quá rườm rà, khó khăn cho việc tái sử dụng lại. Để giải quyết vấn đề này, chúng ta sẽ cùng nhau tìm hiểu về **hàm trong C#.**

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- CẤU TRÚC CƠ BẢN MỘT CHƯƠNG TRÌNH TRONG C# console application
- BIẾN và KIỂU DỮ LIỆU trong C#
- TOÁN TỬ TRONG C#
- CÂU ĐIỀU KIỆN TRONG C#
- CẤU TRÚC CƠ BẢN VÒNG LẶP

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Cấu trúc của hàm trong C#

Cấu trúc của hàm trong C#

Cú pháp khai báo hàm:

> **[Từ khóa 1] [Từ khóa 2] [Từ khóa n] <Kiểu dữ liệu trả về> <Tên hàm>([Parameter]){ }**

Trong đó:

- [Từ khóa 1], [Từ khóa 2], [Từ khóa n] là các từ khóa như: public, static, read only ... và có thể không điền.
- Kiểu dữ liệu trả về như: từ khóa void, hay mọi kiểu dữ liệu như int, long, bool, SinhVien...
- Tên hàm:
  - Là tên gọi của **hàm**.
  - Tên bạn có thể đặt tùy ý nhưng nên đặt tên theo quy tắc đặt tên để có sự đồng bộ ngầm định giữa các lập trình viên và dễ tìm, dễ nhớ (xem lại quy tắc đặt tên ở bài BIẾN TRONG C#).
  - Hãy xem cách khởi tạo hàm giống khởi tạo một biến ở chỗ. Đều cần kiểu dữ liệu và tên. Có thể có các từ khóa. Tên để tái sử dụng hàm ở nơi mong muốn.
- Parameter là tham số truyền vào để sử dụng nội bộ trong hàm. Cấu trúc khởi tạo như một biến bình thường. Có thể không điền.
- Hàm chỉ được khai báo bên trong class.

> **Lưu ý:**
>
> - Mọi hàm đều phải có cặp ngoặc nhọn { } biểu thị là một khối lệnh. Mọi dòng code xử lý của hàm đều được viết bên trong cặp ngoặc nhọn { } này.
> - Không thể khai báo một hàm trong một hàm khác theo cách thông thường.

Một hàm cơ bản hay thấy với cấu trúc bắt buộc phải có trong lập trình C# console hàm Main

```
static void Main(string[] args)
    {
        }
```

Trong đó:

- static là từ khóa static (sẽ tìm hiểu kỹ hơn ở bài sau). Có thể không sử dụng cũng được. Nhưng ở trường hợp hàm Main của console C# thì phải có.
- void là kiểu trả về. Với hàm có kiểu trả về là void thì sẽ không cần từ khóa return trong hàm. Hoặc có nhưng chỉ đơn giản là ghi return;
- Main là tên hàm. Có thể đặt tùy ý. Nhưng ở trường hợp này là bắt buộc phải là Main vì mỗi chương trình console C# đều cần hàm Main.
- string[] args là parameter truyền từ bên ngoài vào để sử dụng hàm. Có thể không có cũng được. nhưng ở trường hợp hàm Main của console C# là bắt buộc phải có. Ở đây có thể thay thế tên args bằng bất cứ tên nào khác như đặt tên một biến bình thường.

Hàm void

Hàm void là hàm có kiểu trả về là void. Chúng ta cùng xem qua khai báo hàm sau:

```
void Demo()
    {
      // some code
     return;
    }
```

- Vì hàm void (hàm có kiểu trả về là void) thì không cần viết **return**; nên chúng ta có thể bỏ **return**; đi.

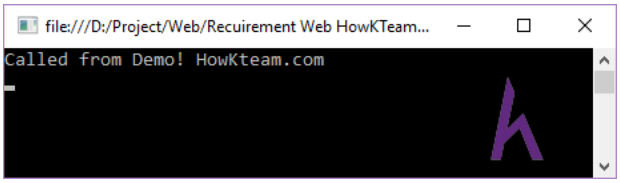
```
void Demo()
    {
      // some code
    }
```

Một lưu ý về sau: vì chúng ta đang viết code trên nền console C#. Bắt buộc phải có hàm Main. Nhưng hàm Main lại có từ khóa static. Nên để trong hàm Main có thể sử dụng các hàm mà ta viết ra thì các hàm đó cũng phải có từ khóa static.

:

```
static void Main(string[] args)
        {
            // Gọi lại hàm để sử dụng
            Demo();
            Console.ReadKey();
        }

        static void Demo()
        {
            Console.WriteLine("Called from Demo! HowKteam.com");
        }
```

- Kết quả vẫn xuất ra dòng chữ "Called from Demo! HowKteam.com" như được viết bên trong hàm Main. Nhưng thật sự nó đã được gọi từ hàm Demo.

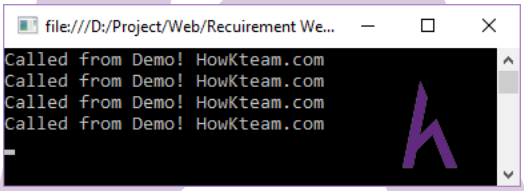


- Khi sử dụng hàm ta sẽ gọi lại tên hàm kèm theo dấu () biểu thị đó là một hàm. Sau này nếu có parameter thì sẽ thêm giá trị vào bên trong dấu ().
- Chúng ta có thể gọi lại nhiều lần và có thể thấy code chúng ta viết rất rõ ràng và rất dễ tái sử dụng.

:

```
static void Main(string[] args)
        {
            // Gọi lại hàm nhiều lần
            // dòng chữ Called from Demo! HowKteam.com cũng được in ra nhiều lần
            Demo();
            Demo();
            Demo();
            Demo();
            Console.ReadKey();
        }

        static void Demo()
        {
            Console.WriteLine("Called from Demo! HowKteam.com");
        }
```

Nhiều dòng chữ "Called from Demo! HowKteam.com" được in ra màn hình



Hàm có kiểu trả về khác void

Với hàm có kiểu trả về khác void. Trong thân hàm bắt buộc phải có dòng **return** <Giá trị trả về>;

Giá trị trả về phải có kiểu dữ liệu tương ứng với Kiểu dữ liệu trả về khi khai báo hàm.

:

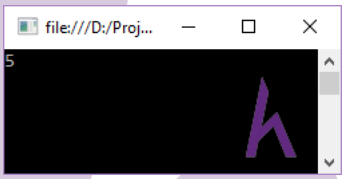
```
  /// <summary>
        /// Hàm trả về giá trị số nguyên 5 thông qua tên hàm
        /// Lưu ý giá trị trả về phải cùng kiểu dữ liệu với kiểu trả về của hàm
        /// Ở đây là kiểu int
        /// </summary>
        /// <returns></returns>
        static int ReturnANumber()
        {
            // bắt buộc phải có cấu trúc return trong thân hàm
            return 5;
        }
```

Chúng ta có thể sử dụng hàm này bình thường. Và có thêm một lợi thế là có thể lấy giá trị trả về của hàm thông qua lời gọi tên hàm.

:

```
static void Main(string[] args)
        {
            Console.WriteLine(ReturnANumber());
            Console.ReadKey();
        }

        /// <summary>
        /// Hàm trả về giá trị số nguyên 5 thông qua tên hàm
        /// Lưu ý giá trị trả về phải cùng kiểu dữ liệu với kiểu trả về của hàm
        /// Ở đây là kiểu int
        /// </summary>
        /// <returns></returns>

        static int ReturnANumber()
        {
            // bắt buộc phải có cấu trúc return trong thân hàm
            return 5;
        }
```

Kết quả số 5 xuất hiện trên màn hình

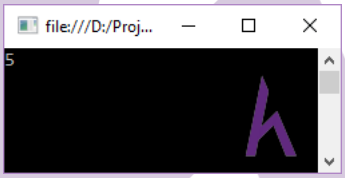


Chúng ta có thể gọi hàm trong hàm như sau: (để hiểu về trình tự gọi hàm các bạn có thể xem lại phần ví dụ của bài NHẬP XUẤT CONSOLE CƠ BẢN TRONG C#)

:

```
static void Main(string[] args)
        {
            Demo();
            Console.ReadKey();
        }


        /// <summary>
        /// Hàm trả về giá trị số nguyên 5 thông qua tên hàm
        /// Lưu ý giá trị trả về phải cùng kiểu dữ liệu với kiểu trả về của hàm
        /// Ở đây là kiểu int
        /// </summary>
        /// <returns></returns>

        static int ReturnANumber()
        {
            // bắt buộc phải có cấu trúc return trong thân hàm
            return 5;
        }

        /// <summary>
        /// in ra màn hình một giá trị nào đó
        /// </summary>

        static void Demo()
        {
         // trong hàm Demo gọi lại hàm ReturnANumber
            Console.WriteLine(ReturnANumber());
        }
```

Kết quả vẫn là số 5 được in ra màn hình



Parametter

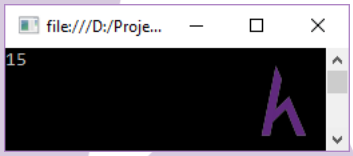
Chúng ta đã biết cách khởi tạo và sử dụng một hàm. Vậy giờ có một yêu cầu như sau: Viết hàm tính tổng 2 số nguyên.

- Chúng ta có thể sử dụng biến toàn cục (sẽ được nói rõ ở bài sau) để giải quyết:

:

```
static void Main(string[] args)
        {
            Console.WriteLine(SumTwoNumber());
            Console.ReadKey();
        }


        /// <summary>
        ///  hai biến firstNumber và secondNumber hiện là biến toàn cục của các hàm nằm bên trong class Program nhưng lại là biến
cục bộ của class Program
        ///  Cần có từ khóa static vì các hàm sử dụng nó đều có từ khóa static
        /// </summary>

        static int firstNumber = 5;
        static int secondNumber = 10;


        /// <summary>
        /// hàm trả ra kết quả tổng của 2 số firstNumber và secondNumber
        /// </summary>
        /// <returns></returns>

        static int SumTwoNumber()
        {
            return firstNumber + secondNumber;
        }
```

Kết quả màn hình xuất ra giá trị tổng của hai biến **firstNumber** và **secondNumber**. 5 + 10 = 15



Nhưng khi dùng phương pháp như vậy rõ là khá phiền phức khi muốn in ra màn hình tổng của hai số một cách linh hoạt. Hay muốn thực hiện tính tổng hai số nhiều lần. Để tạo sự linh hoạt cho hàm thì chúng ta sẽ tìm hiểu thêm về parameter:

- Có thể hiểu đơn giản parametter là
  - Tập hợp một hay nhiều biến chứa các giá trị cần thiết để thao tác trong hàm.
  - Các giá trị của các biến này là những giá trị mà người dùng truyền vào khi gọi hàm đó.
  - Khai báo một parametter cũng giống như khai báo biến (xem lại bài BIẾN TRONG C#). Khi khai báo nhiều parametter thì các khai báo phải cách nhau bởi dấu ","

Trở lại bài toán trên. Chúng ta muốn 2 số cần tính tổng này là 2 số do người dùng quyết định, 2 số này không cố định. Vì thế ta nảy sinh ý tưởng:

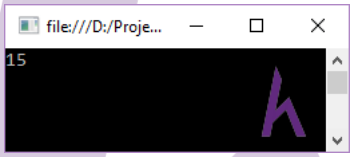
- Cho người dùng truyền vào 2 số họ muốn tính tổng vào 2 biến.
- Từ đó ta chỉ cần tính tổng giá trị 2 biến đó rồi trả kết quả về cho người dùng.

Các bạn xem qua ví dụ sau:

:

```
static void Main(string[] args)
        {

            // khi sử dụng hàm phải truyền đúng số lượng, thứ tự parameter vào như khai báo của hàm
            // đồng thời kiểu dữ liệu truyền vào của parameter phải trùng khớp với khai báo của hàm

            Console.WriteLine(SumTwoNumber(5, 10));
            Console.ReadKey();
        }


        /// <summary>
        /// hàm trả ra kết quả tổng của 2 số firstNumber và secondNumber
        /// </summary>

        /// <param name="firstNumber"></param>
        /// <param name="secondNumber"></param>
        /// <returns></returns>

        static int SumTwoNumber(int firstNumber, int secondNumber)
        {
            return firstNumber + secondNumber;
        }
```

Kết quả màn hình vẫn in ra giá trị 15.



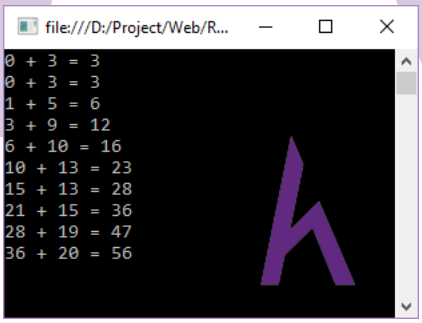
- Các khai báo int **firstNumber**, int **secondNumber** là các khai báo parametter. Với khai báo này ta hiểu rằng muốn sử dụng hàm này thì cần truyền vào 2 giá trị kiểu int.
- Các parametter được xem như các biến cục bộ có phạm vị sử dụng trong hàm (biến cục bộ sẽ được trình bày ở bài sau).
- Các parametter được khởi tạo ngay khi gọi hàm và được hủy khi kết thúc gọi hàm.
- Số lượng parameter là không giới hạn.
- Khi sử dụng hàm phải truyền vào đủ và đúng parameter. (Đủ số lượng, đúng kiểu dữ liệu và đúng thứ tự như khai báo)
- Có thể khai báo các parameter với các kiểu dữ liệu khác nhau.
- Hàm sử dụng sẽ tạo ra các bản sao của parameter truyền vào trên RAM. Sau đó dùng những bản sao đó để xử lý dữ liệu. Cho nên kết thúc lời gọi hàm giá trị của các parameter sẽ không bị thay đổi.

Cùng xét thêm một ví dụ nữa nào:

:

```
static void Main(string[] args)
        {
            int firstNum = 0;
            int secondNum = 3;


            // in ra màn hình 10 lần tổng 2 số
            for (int count = 0; count < 10; count++)
            {
                // in ra màn hình tổng của 2 số
                PrintSumTwoNumber(firstNum, secondNum);


                // tính toán để tạo ra 2 số mới. Không quan trọng lắm
                firstNum += count;
                secondNum += count * 2 % 5;
            }

            Console.ReadKey();
        }


        /// <summary>
        /// In ra màn hình tổng của 2 số
        /// </summary>
        /// <param name="firstNumber"></param>
        /// <param name="secondNumber"></param>

        static void PrintSumTwoNumber(int firstNumber, int secondNumber)
        {
            Console.WriteLine("{0} + {1} = {2}",firstNumber, secondNumber, SumTwoNumber(firstNumber, secondNumber));
        }

        /// <summary>
        /// hàm trả ra kết quả tổng của 2 số firstNumber và secondNumber
        /// </summary>
        /// <param name="firstNumber"></param>
        /// <param name="secondNumber"></param>
        /// <returns></returns>

        static int SumTwoNumber(int firstNumber, int secondNumber)
        {
            return firstNumber + secondNumber;
        }
```

Kết quả màn hình xuất ra 10 giá trị tổng của 2 số

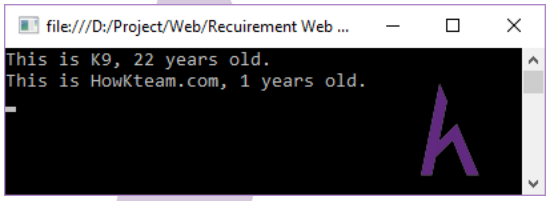


- Ví dụ về các parameter với kiểu dữ liệu khác nhau

:

```
static void Main(string[] args)
        {
            PrintSomeThing("K9", 22);
            PrintSomeThing("HowKteam.com", 1);
            Console.ReadKey();
        }


        static void PrintSomeThing(string name, int age)
        {
            // in ra màn hình tên và tuổi được truyền vào
            Console.WriteLine("This is {0}, {1} years old.", name, age);
        }
```

Kết quả khi chạy chương trình:



- Nếu không sử dụng parametter trong hàm thì không nên khai báo parametter để tránh lãng phí bộ nhớ!

Kết luận

Qua bài này chúng ta đã nắm được hàm là gì, cách sử dụng hàm.

Bài sau chúng ta sẽ tìm hiểu về **BIẾN TOÀN CỤC VÀ BIẾN CỤC BỘ TRONG C#**.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".