

LESSON 06

I. CẤU TRÚC CƠ BẢN CỦA HÀM TRONG C#

❖ Định nghĩa

Khi mình muốn thực thi một đoạn code nào đó nhiều lần, thay vì phải copy đi copy lại đoạn code đó nhiều lần, dẫn đến chương trình chúng ta bị trùng lặp code rất nhiều, trong c# có function cho phép chúng ta thực thi đoạn code nào đó nhiều lần mà không cần phải copy lại code, mà chỉ cần gọi tên hàm.

Vậy cách sử dụng hàm (*function*) trong c# như thế nào? Có bao nhiêu cách để truyền tham số vào hàm? Chúng ta cùng tìm hiểu trong nội dung tiếp theo.

Là một trong những cú pháp vòng lặp đã đề cập trong Lesson04 và cũng rất quan trọng trong lập trình C#.

➤ **Cú pháp:** Hàm (function) trong C# dùng để thực thi một khối lệnh nào đó.

```
[Từ khóa 1] [...] [Từ khóa n] <Kiểu dữ liệu trả về> <Tên hàm>([Parameter])
{
    //Khối lệnh thực hiện trong hàm
}
```

➤ **Trong đó:**

- [Từ khóa 1], [...], [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** 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.
- **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**.
- **Điều kiện lặp** là một biểu thức logic bắt buộc phải có với kết quả trả về bắt buộc là **true** hoặc **false**.

➤ **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.

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

```
0 references
static void Main(string[] args)
{
    int countLoop = 0;
    int countLoopTime = 0;

    int valueNum = 10;
    int loopTime = 5;

    // Vẽ từ trên xuống LoopTime lần
    while (countLoopTime < loopTime)
    {
        countLoop = 0;
        // vẽ từ trái qua valueNum lần
        while (countLoop < valueNum)
        {
            Console.Write("{0,8}", countLoop);
            countLoop++;
        }
        // Mỗi khi hoàn thành một vòng lặp nhỏ thì lại xuống dòng chuẩn vị vẽ lần tiếp theo
        Console.WriteLine();
        countLoopTime++;
    }
    Console.ReadKey();
}
```

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 có kiểu trả về là VOID

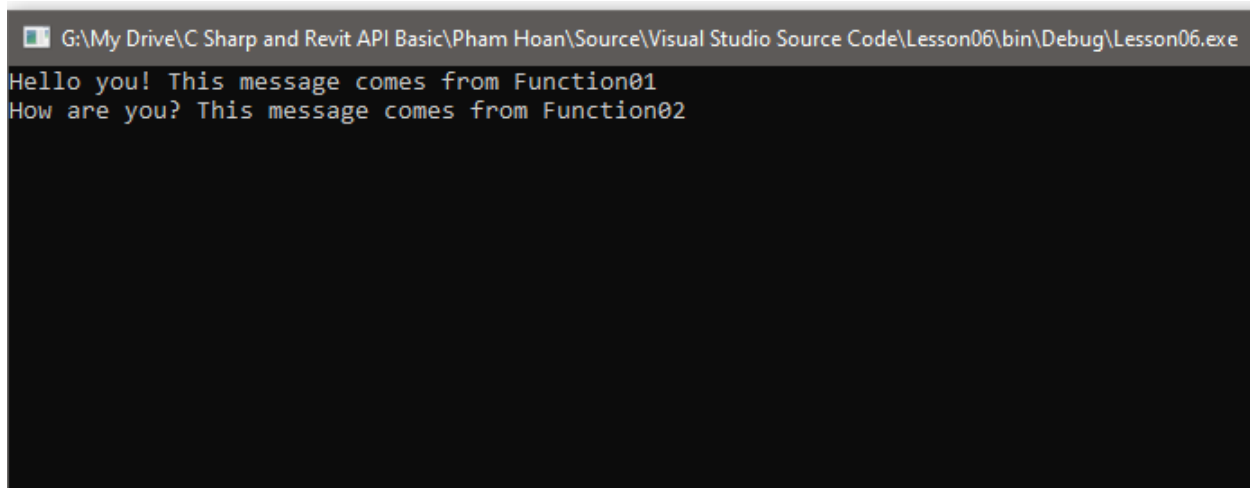
Với hàm có kiểu trả về là **void** thì sẽ không cần từ khóa **return** trong hàm. Hoặc có nhưng chỉ đơn giản là ghi **return**;

➤ Ví dụ:

```
0 references
static void Main(string[] args)
{
    MyFirstFunction();
    MySecondFunction();
    Console.ReadKey();
}

// Vì hàm Main có static nên các hàm con sẽ phải có static (Đặc tính này sẽ được học trong các bài sau)
1 reference
static void MyFirstFunction()
{
    Console.WriteLine("Hello you! This message comes from Function01");
    return;
}

1 reference
static void MySecondFunction()
{
    Console.WriteLine("How are you? This message comes from Function02");
}
```



```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson06\bin\Debug\Lesson06.exe
Hello you! This message comes from Function01
How are you? This message comes from Function02
```

- 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.

```
0 references
static void Main(string[] args)
{
    MyFirstFunction();
    MySecondFunction();

    MyFirstFunction();
    MySecondFunction();

    MyFirstFunction();
    MySecondFunction();

    MyFirstFunction();
    MySecondFunction();
}

G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson06\bin\Debug\Lesson06.exe
Hello you! This message comes from Function01
How are you? This message comes from Function02
Hello you! This message comes from Function01
How are you? This message comes from Function02
Hello you! This message comes from Function01
How are you? This message comes from Function02
Hello you! This message comes from Function01
How are you? This message comes from Function02
```

❖ 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 có kiểu trả về khác void
/// </summary>
/// <param name="args"></param>
1 reference
static int GetCurrentYear()
{
    int year;
    year = DateTime.Now.Year;

    return year;
}

0 references
static void Main(string[] args)
{
    int current = GetCurrentYear();
    Console.WriteLine("Nam hiện tại là: " + current);
    Console.ReadKey();
}
```

❖ Parameter

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:

```
/// **** PARAMETERS ****/  
///  
static int a, b;  
0 references  
static void Main(string[] args)  
{  
    a = 5;  
    b = 6;  
    int c = Sum();  
    Console.WriteLine("Tong 2 so la: " + c);  
}  
  
1 reference  
static int Sum()  
{  
    return a + b;  
}  
}
```

Kết quả màn hình xuất ra giá trị tổng của hai biến **a** và **b**: $5 + 6 = 11$

G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson06\bin\Debug\Lesson06.exe

Tong 2 so la: 11

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 **parameter** 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 **parameter** cũng giống như khai báo biến (xem lại bài **BIÊN TRONG C#**). Khi khai báo nhiều **parameter** 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.

```
static void Main(string[] args)
{
    a = 5;
    b = 6;
    int c = Sum();
    Console.WriteLine("Tổng 2 số là: " + c);
    Console.ReadKey();

    // CÁCH 2
    int d = AnotherSum(a, b);
    Console.WriteLine("Tổng 2 số là: " + d);
    Console.ReadKey();
}

1 reference
static int Sum(...)

1 reference
static int AnotherSum(int firstNumber, int secondNumber)
{
    return firstNumber + secondNumber;
}
```

➤ Kết luận:

- ✓ Các khai báo `int firstNumber`, `int secondNumber` là các khai báo **parameter**. Với khai báo này ta hiểu rằng muốn sử dụng hàm này thì cần truyền vào 2 giá trị kiểu `int`.
- ✓ Các **parameter** được xem như các biến cục bộ có phạm vi sử dụng trong hàm (biến cục bộ sẽ được trình bày ở bài sau).
- ✓ Các **parameter** 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:

```
0 references
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++)
    {
        PrintSumTwoNumber(firstNum, secondNum);
        // tính toán để tạo ra 2 số mới. Không quan trọng lắm
        firstNum += count;
        secondNum += count * 2 % 5;
    }
    Console.ReadKey();
}

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

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

```
G:\My Drive\C Sharp and Revit API Basic\Pham Hoan\Source\Visual Studio Source Code\Lesson06\bin\Debug\Lesson06.exe
0 + 3 = 3
0 + 3 = 3
1 + 5 = 6
3 + 9 = 12
6 + 10 = 16
10 + 13 = 23
15 + 13 = 28
21 + 15 = 36
28 + 19 = 47
36 + 20 = 56
-
```

II. BIẾN TOÀN CỤC, BIẾN CỤC BỘ TRONG C#

```
// ***** BIẾN TOÀN CỤC & BIẾN CỤC BỘ *****
// biến toàn cục của các hàm nằm trong class Program
static int value = 5;
0 references
static void Main(string[] args)
{
    // in ra màn hình biến toàn cục
    Console.WriteLine(value);
    // thay đổi giá trị của value
    value = 10;
    // kết quả gọi hàm này sẽ không thay đổi vì ưu tiên biến cục bộ hơn
    PrintSomething();
    Console.ReadKey();
}

1 reference
static void PrintSomething()
{
    int value = 9;
    Console.WriteLine(value);
}
```

BÀI TẬP

1. Viết chương trình C# để tạo một hàm để tính giá trị của x^y . Ví dụ:

```
Nhập cơ số: 3
Nhập lũy thừa: 4
Giá trị của 34 = 81
```

2. Viết chương trình nhập vào 1 mảng và xuất ra phần tử max nhất.
3. Tương tự bài 2 nhưng xuất ra mảng đã sắp xếp từ nhỏ tới lớn

4. Cho số nguyên dương n được nhập từ bàn phím, bạn hãy viết phương thức về $n!$ (n giai thừa).

III. BIẾN TOÀN CỤC VÀ BIẾN CỤC BỘ TRONG C#

Biến toàn cục là biến được khai báo ở phân cấp cao hơn vị trí đang xác định.

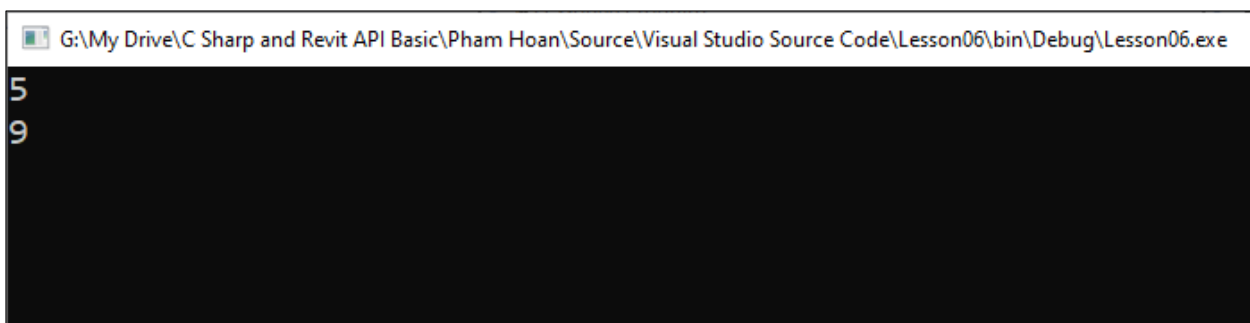
Biến cục bộ là biến được khai báo ở cùng phân cấp tại vị trí đang xác định.

Vòng đời của biến toàn cục và biến cục bộ bắt đầu khi khối lệnh chứa nó bắt đầu (khối lệnh bắt đầu bằng dấu “{”) và kết thúc khi khối lệnh chứa nó kết thúc (khối lệnh kết thúc bằng dấu “}”).

Biến cục bộ được ưu tiên sử dụng hơn **biến toàn cục** trong trường hợp 2 biến này trùng tên.

```
// ***** BIẾN TOÀN CỤC & BIẾN CỤC BỘ *****
// biến toàn cục của các hàm nằm trong class Program
static int value = 5;
0 references
static void Main(string[] args)
{
    // in ra màn hình biến toàn cục
    Console.WriteLine(value);
    // thay đổi giá trị của value
    value = 10;
    // kết quả gọi hàm này sẽ không thay đổi vì ưu tiên biến cục bộ hơn
    PrintSomething();
    Console.ReadKey();
}

1 reference
static void PrintSomething()
{
    int value = 9;
    Console.WriteLine(value);
}
```



Lưu ý:

- **Parameter** chính là một biến cục bộ.
- **Biến cục bộ** có phạm vi sử dụng bên trong cặp dấu ngoặc nhọn { }