

# Bài: Mảng nhiều chiều trong lập trình C# cơ bản

Xem bài học trên website để ủng hộ Kteam: [Mảng nhiều chiều trong lập trình C# cơ bản](#)

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 [MẢNG HAI CHIỀU TRONG C#](#). Hôm nay chúng ta sẽ cùng tìm hiểu về **mảng nhiều chiều trong C#**.

## Nội dung

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

- [MẢNG 1 CHIỀU TRONG C#](#)
- [MẢNG 2 CHIỀU TRONG C#](#)

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

- Mảng 3 chiều trong C#
- Mảng jagged trong C#
- Lớp Array trong C#

## Mảng nhiều chiều trong C#

Ngoài [mảng 1 chiều](#) và [mảng 2 chiều](#) ra thì C# còn hỗ trợ chúng ta khai báo và sử dụng mảng 3 chiều, 4 chiều thậm chí là n chiều tùy vào từng yêu cầu của bài toán.

Trong phạm vi bài viết này mình chỉ trình bày về mảng 3 chiều, còn về mảng 4 chiều trở đi thì các bạn có thể tự nghiên cứu thêm.

Về khái niệm của mảng cũng như các đặc trưng cơ bản của mảng mình đã trình bày trong bài [MẢNG 1 CHIỀU TRONG C#](#) rồi nên ở đây mình sẽ không nhắc lại mà đi trực tiếp vào vấn đề luôn.

## Mảng 3 chiều trong C#

Nếu như [mảng 2 chiều](#) được hình dung như là một ma trận  $M \times N$  thì mảng 3 chiều được hình dung như một hình hộp chữ nhật có kích thước  $M \times N \times P$ .

Cú pháp:

```
<kiểu dữ liệu> [ , , ] <tên mảng>;
```

Trong đó:

- **<kiểu dữ liệu>** là kiểu dữ liệu của các phần tử trong mảng.
- Cặp dấu [ , , ] là ký hiệu cho khai báo mảng 3 chiều.
- **<tên mảng>** là tên của mảng, cách đặt tên mảng cũng như cách đặt tên biến (quy tắc đặt tên biến đã trình bày trong [BIẾN TRONG C#](#)).

Để sử dụng được mảng ta phải khởi tạo giá trị hoặc cấp phát vùng nhớ cho mảng.

## Cấp phát vùng nhớ

Được thực hiện tương tự như [mảng 2 chiều](#) nhưng cần chỉ ra 3 chỉ số cho mảng.

**Ví dụ:**

**C#:**

```
/*
 * Khai báo mảng 3 chiều kiểu string và có tên là Kteam.
 * Sau đó thực hiện cấp phát vùng nhớ với 3 chỉ số lần lượt là 2, 2, 3.
 */

string[ , , ] Kteam = new string[2, 2, 3];
```

Sau khi mảng được cấp phát vùng nhớ thì các phần tử trong mảng sẽ mang giá trị mặc định:


- Đối với số nguyên là 0
- Đối với số thực là 0.0
- Đối với kiểu ký tự là " (ký tự rỗng)
- Đối với kiểu tham chiếu là **null**

Chúng ta có thể khởi tạo giá trị khác mà chúng ta mong muốn ngay khi cấp phát vùng nhớ bằng cú pháp sau:

**Khởi tạo 1 mảng 1 chiều mà mỗi phần tử là một mảng 2 chiều**

```
<kiểu dữ liệu>[ , , ] <tên mảng> = new <kiểu dữ liệu>[ , , ]
{
    {
        { <giá trị tại vị trí (0, 0, 0)>, ..., <giá trị tại vị trí (0, 0, p)> },
        ...
        { <giá trị tại vị trí (0, n, 0)>, ..., <giá trị tại vị trí (0, n, p)> }
    }
    ...
    {
        { <giá trị tại vị trí (m, 0, 0)>, ..., <giá trị tại vị trí (m, 0, p)> },
        ...
        { <giá trị tại vị trí (m, n, 0)>, ..., <giá trị tại vị trí (m, n, p)> }
    }
};
```

**Khởi tạo 1 mảng 2 chiều**



- Chúng ta có thể xem mảng 3 chiều là một mảng 1 chiều mà mỗi phần tử là một mảng 2 chiều. Khi đó việc khởi tạo sẽ là:
  - Khởi tạo một mảng 1 chiều.
  - Ở mỗi phần tử ta khởi tạo một mảng 2 chiều.
- Các giá trị khởi tạo nằm trong cặp dấu ngoặc nhọn {} và cách nhau bởi dấu phẩy.
- Chúng ta không cần cung cấp các chỉ số tối đa mà trình biên dịch sẽ tự đếm số lượng phần tử bạn đã khởi tạo ở mỗi chiều và xem nó là chỉ số tối đa cho từng chiều.

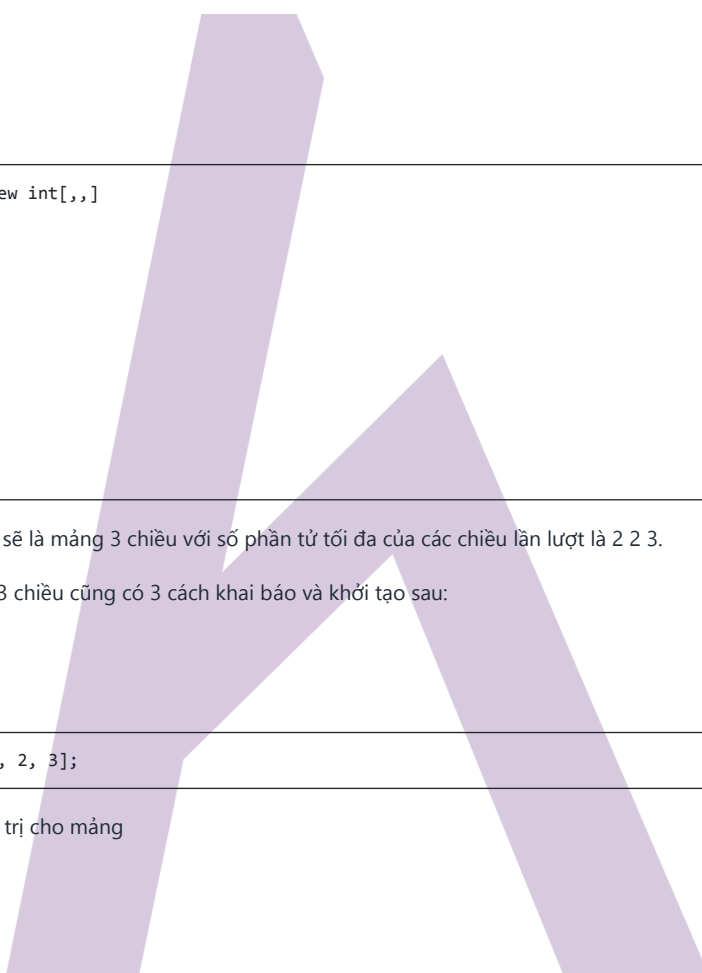
## Khởi tạo giá trị

Việc khởi tạo giá trị hoàn toàn tương tự như phần trên mình đã trình bày.

**Cú pháp:**

`<kiểu dữ liệu>[ , , ] <tên mảng> =`  
`{`  
`{`  
`{ <giá trị tại vị trí (0, 0, 0)>, ..., <giá trị tại vị trí (0, 0, p)> },`  
`...`  
`{ <giá trị tại vị trí (0, n, 0)>, ..., <giá trị tại vị trí (0, n, p)> }`  
`}`  
`...`  
`{`  
`{ <giá trị tại vị trí (m, 0, 0)>, ..., <giá trị tại vị trí (m, 0, p)> },`  
`...`  
`{ <giá trị tại vị trí (m, n, 0)>, ..., <giá trị tại vị trí (m, n, p)> }`  
`}`  
`}`  
`};`

**Khởi tạo 1 mảng 1 chiều mà mỗi phần tử là một mảng 2 chiều**  
**Khởi tạo 1 mảng 2 chiều**


**Ví dụ:****C#:**

```
int[,,] Mang3Chieu = new int[,,]
{
    {
        {1, 2, 3},
        {4, 5, 6}
    },
    {
        {7, 8, 9},
        {10, 11, 12}
    }
};
```

- Với khai báo trên thì Mang3Chieu sẽ là mảng 3 chiều với số phần tử tối đa của các chiều lần lượt là 2 2 3.

Tóm lại, cũng như mảng 2 chiều, mảng 3 chiều cũng có 3 cách khai báo và khởi tạo sau:

- Khai báo và cấp phát vùng nhớ

**C#:**

```
string[ , , ] Array = new string[2, 2, 3];
```

- Khai báo, cấp phát và khởi tạo giá trị cho mảng


```

<kiểu dữ liệu>[ , , ] <tên mảng> =
{
    {
        { <giá trị tại vị trí (0, 0, 0)>, ..., <giá trị tại vị trí (0, 0, p)> },
        ...
        { <giá trị tại vị trí (0, n, 0)>, ..., <giá trị tại vị trí (0, n, p)> }
    }
    ...
    {
        { <giá trị tại vị trí (m, 0, 0)>, ..., <giá trị tại vị trí (m, 0, p)> },
        ...
        { <giá trị tại vị trí (m, n, 0)>, ..., <giá trị tại vị trí (m, n, p)> }
    }
};

```

**Khởi tạo 1 mảng 1 chiều mà mỗi phần tử là một mảng 2 chiều**

**Khởi tạo 1 mảng 2 chiều**



- Khởi tạo giá trị cho mảng


```

<kiểu dữ liệu>[ , , ] <tên mảng> =
{
    {
        { <giá trị tại vị trí (0, 0, 0)>, ..., <giá trị tại vị trí (0, 0, p)> },
        ...
        { <giá trị tại vị trí (0, n, 0)>, ..., <giá trị tại vị trí (0, n, p)> }
    }
    ...
    {
        { <giá trị tại vị trí (m, 0, 0)>, ..., <giá trị tại vị trí (m, 0, p)> },
        ...
        { <giá trị tại vị trí (m, n, 0)>, ..., <giá trị tại vị trí (m, n, p)> }
    }
};

```

**Khởi tạo 1 mảng 1 chiều mà mỗi phần tử là một mảng 2 chiều**

**Khởi tạo 1 mảng 2 chiều**



## Sử dụng mảng 3 chiều:

Tương tự như mảng 2 chiều, kiểu mảng 3 chiều cũng có thể dùng làm:

- Kiểu dữ liệu cho biến.
- Kiểu trả về cho hàm.
- Tham số truyền vào cho hàm.

Các phần tử của mảng được truy xuất thông qua 3 chỉ số ngăn cách nhau bởi dấu phẩy và nằm trong cặp dấu [].

**C#:**

```
// Khai báo, cấp phát và khởi tạo mảng 3 chiều kiểu int và tên là Mang3Chieu
int[, ,] Mang3Chieu = new int[, ,]
{
    {
        {1, 2, 3},
        {4, 5, 6}
    },
    {
        {7, 8, 9},
        {10, 11, 12}
    }
};

/*
 * Truy xuất đến phần tử có các chỉ số lần lượt là 1 1 2
 */
Console.WriteLine(Mang3Chieu[1, 1, 2]);
```

Tương tự như mảng 2 chiều, mảng 3 chiều cũng có một số thuộc tính và phương thức đặc trưng như mảng 2 chiều:

Tên thuộc tính hoặc phương thức	Ý nghĩa
Length	Thuộc tính trả về số nguyên kiểu <b>int</b> là số phần tử tối đa của mảng (số phần tử của mảng 3 chiều là tích 3 chỉ số của mảng)
LongLength	Tương tự như thuộc tính Length nhưng trả về số nguyên kiểu <b>long</b>
GetLength(<số chiều>)	Trả về số nguyên kiểu <b>int</b> là số phần tử trong chiều đã xác định. Lưu ý chiều của mảng là các số nguyên và được đánh số từ 0. Đối với mảng 3 chiều thì GetLength(0) là độ dài chiều đầu tiên và GetLength(1) là độ dài chiều thứ 2 và GetLength(2) là độ dài chiều thứ 3.
GetLongLength(<số chiều>)	Tương tự GetLength nhưng trả về số nguyên kiểu <b>long</b>
Rank	Thuộc tính trả về số nguyên <b>int</b> là số chiều của mảng
Clone()	Thực hiện copy giá trị của mảng ra một vùng nhớ mới (phép gán thông thường thì 2 đối tượng sẽ dùng chung vùng nhớ rất nguy hiểm vì đối tượng này thay đổi dẫn đến đối tượng kia cũng thay đổi)

Còn rất nhiều thuộc tính và phương thức khác, mình chỉ giới thiệu một số cái hay dùng còn lại các bạn có thể tự khám phá.

## Cách duyệt mảng 3 chiều:

- **Ý tưởng:**
  - Tương tự như ý tưởng duyệt mảng 2 chiều (đã trình bày trong bài [MẢNG 2 CHIỀU TRONG C#](#))
  - Nhưng do mảng 3 chiều có 3 chỉ số nên chúng ta cần 3 vòng lặp lồng vào nhau.
- **Ví dụ:**

C#:

```
int[, ,] IntArray = new int[3, 9, 10];

/*
 * Sử dụng 3 vòng for lồng vào nhau để duyệt mảng 3 chiều
 * Vòng lặp đầu tiên là vòng lặp duyệt các phần tử của chiều đầu tiên.
 * Vòng lặp thứ 2 là vòng lặp duyệt các phần tử của chiều thứ 2.
 * Vòng lặp thứ 3 là vòng lặp duyệt các phần tử của chiều thứ 3.
 */

for (int i = 0; i < IntArray.GetLength(0); i++)
{
    for (int j = 0; j < IntArray.GetLength(1); j++)
    {
        for (int k = 0; k < IntArray.GetLength(2); k++)
        {
            /*
             * Với cách duyệt này thì IntArray[i, j, k] sẽ là phần tử hiện tại mình đang xét
             * Code xử lý sẽ viết ở đây
             */
        }
    }
}
```

Cách duyệt này sẽ duyệt tuần tự các chiều trong mảng 3 chiều, ở mỗi chiều sẽ duyệt từ đầu đến cuối. Bạn hoàn toàn có thể duyệt theo ý mình bằng cách thay đổi giá trị trong vòng lặp.

Vì mảng 3 chiều cũng ít được sử dụng nên mình chỉ giới thiệu qua thôi chứ không đi quá chi tiết. Với kiến thức cơ bản mình đã cung cấp các bạn hoàn toàn có thể tự tìm hiểu thêm khi cần thiết.

## Mảng jagged trong C#

**Mảng jagged** (hay còn gọi là mảng lờm chờm) là một mảng của các mảng. Nghe có vẻ giống [mảng 2 chiều](#) đã học nhưng thực ra là rất khác.

Đặc điểm của mảng jagged cũng là điểm khác biệt giữa mảng này với mảng 2 chiều:

- Số phần tử của chiều thứ 2 có thể khác nhau (đối với mảng 2 chiều là bằng nhau).
- Các ô nhớ được cấp phát có thể không nằm liền kề nhau (đối với mảng 2 chiều là các ô nhớ sẽ được cấp phát liền kề nhau).
- Bản chất vẫn là mảng 1 chiều nhưng các phần tử có thể chứa 1 mảng khác.

Ưu điểm lớn nhất của mảng jagged là tiết kiệm bộ nhớ. Bởi vì khi mình cần xài bao nhiêu thì mình cấp phát bấy nhiêu nên sẽ không bị thừa một ô nhớ nào.

## Khai báo mảng jagged

Cú pháp:

```
<kiểu dữ liệu> [ ][ ] <tên mảng>;
```

Trong đó:

- <kiểu dữ liệu> là kiểu dữ liệu của các phần tử trong mảng.
- Cặp dấu [ ][ ] là ký hiệu cho khai báo mảng jagged.
- <tên mảng> là tên của mảng, cách đặt tên mảng cũng như cách đặt tên biến (quy tắc đặt tên biến đã trình bày trong [BIẾN TRONG C#](#)).

Ví dụ:

C#:

```
int [][] JaggedArray;
```

## Cấp phát vùng nhớ

Không giống như [mảng 2 chiều](#), mảng jagged không cho phép cấp phát cùng lúc số dòng và số cột mà phải cấp phát số dòng trước sau đó ứng với mỗi dòng ta cấp phát số cột theo ý muốn.

Để hiểu hơn về cách cấp phát vùng nhớ cho mảng jagged thì ta xem qua ví dụ sau:

C#:

```
int [][] JaggedArray = new int[3][];  
  
JaggedArray[0] = new int[3];  
  
JaggedArray[1] = new int[9];  
  
JaggedArray[2] = new int[10];
```

Từ ví dụ ta có thể thấy:

- Vì mảng jagged là mảng 1 chiều nên đầu tiên cần cấp phát số phần tử cho [mảng 1 chiều](#).
- Sau đó ứng với mỗi phần tử của [mảng 1 chiều](#) đó ta cấp phát 1 mảng 1 chiều nữa.
- Chính vì điểm này nên ta có thể linh động chọn số phần tử phù hợp cho từng dòng.

## Khởi tạo giá trị

Cách khởi tạo giá trị của mảng jagged cũng có chút khác so với [mảng 2 chiều](#). Ta xem qua ví dụ sau:

C#:

```
int[][] JArray =  
{  
    new int[] {1, 2, 3},  
    new int[] {3, 4, 5, 6, 7, 8, 9}  
};
```

Từ ví dụ có thể thấy là:

- Không thể khởi tạo trực tiếp giá trị từng vị trí như mảng 2 chiều được.
- Ứng với mỗi dòng ta phải cấp phát vùng nhớ rồi mới khởi tạo giá trị được.

## Sử dụng mảng jagged

Khi sử dụng mảng jagged các bạn cần lưu ý:

- Đây không phải là mảng 2 chiều nên cách truy xuất đến 1 phần tử có chút thay đổi trong cú pháp. Đó là sử dụng **<tên mảng>[i][j]** thay vì **<tên mảng>[i, j]**.
- Lúc này i, j không được xem là chỉ số dòng và chỉ số cột nữa mà có thể hiểu i là chỉ số chỉ vị trí phần tử trong mảng 1 chiều (ở đây mỗi phần tử trong mảng 1 chiều này là 1 mảng 1 chiều), còn j là chỉ số phần tử của mảng 1 chiều nằm tại vị trí i trên.
- Chỉ có thể sử dụng phương thức **GetLength(0)** chứ không thể sử dụng **GetLength(1)** vì mảng jagged bản chất vẫn là mảng 1 chiều.
- Nếu ta truy xuất **<tên mảng>[i]** thì có thể xem đây là 1 mảng 1 chiều và có thể thao tác như một mảng 1 chiều bình thường.

## Cách duyệt mảng jagged

Các bạn hoàn toàn có thể duyệt mảng jagged bằng 2 [vòng lặp for](#) lồng vào nhau như mảng 2 chiều. Nhưng chúng ta cần linh hoạt hơn ở vòng lặp thứ 2 để tránh chương trình bị lỗi.

Cùng xét qua ví dụ sau:

**C#:**

```
/*  
  
    * Khai báo 1 mảng jagged tên JArray và có 2 phần tử tương ứng với 2 mảng con.  
  
    * Mảng con thứ nhất là mảng 1 chiều có 3 phần tử.  
  
    * Mảng con thứ hai là mảng 1 chiều có 7 phần tử.  
  
    */  
  
int[][] JArray =  
  
    {  
  
        new int[] {1, 2, 3},  
  
        new int[] {3, 4, 5, 6, 7, 8, 9}  
  
    };  
  
/*  
  
    * Sử dụng 2 vòng lặp lồng vào nhau để duyệt mảng jagged này.  
  
    * Vòng lặp đầu tiên là vòng lặp duyệt số phần tử của mảng 1 chiều.  
  
    * Vì mỗi phần tử là 1 mảng 1 chiều nên vòng lặp thứ 2 là để duyệt mảng 1 chiều tại vị trí tương ứng.  
  
    * Do các mảng 1 chiều có kích thước khác nhau nên ta dùng thuộc tính Length để xác định kích thước.  
  
    */  
  
for (int i = 0; i < JArray.Length; i++)  
  
    {  
  
        for (int j = 0; j < JArray[i].Length; j++)  
  
            {  
  
                /*  
  
                    * JArray[i][j] là cách truy xuất đến mảng 1 chiều thứ i và phần tử thứ j trong mảng đó.  
  
                    */  
  
                Console.WriteLine("\t" + JArray[i][j]);  
  
            }  
  
        Console.WriteLine();  
  
    }  
  
}
```



## Lớp Array trong C#

**Lớp Array** trong C# là lớp cơ sở cho mọi mảng, một mảng bất kỳ đều được kế thừa từ lớp này (khái niệm kế thừa sẽ được trình bày trong bài [TÍNH KẾ THỪA TRONG C#](#)).

Một số thuộc tính trong lớp Array:

Tên thuộc tính	Ý nghĩa
Length	Thuộc tính trả về số nguyên kiểu <b>int</b> là số phần tử tối đa của mảng.
LongLength	Tương tự như thuộc tính Length nhưng trả về số nguyên kiểu <b>long</b>
Rank	Thuộc tính trả về số nguyên <b>int</b> là số chiều của mảng

Một số phương thức trong lớp Array:

Tên phương thức	Ý nghĩa
Clear()	Thiết lập lại giá trị mặc định cho tất cả các phần tử trong mảng.
GetLength(<số chiều>)	Trả về số nguyên kiểu <b>int</b> là số phần tử trong chiều đã xác định. Lưu ý chiều của mảng là các số nguyên và được đánh số từ 0.
GetLongLength(<số chiều>)	Tương tự GetLength nhưng trả về số nguyên kiểu <b>long</b>
GetValue(<vị trí>)	Trả về giá trị 1 phần tử của mảng tại vị trí truyền vào. Nếu là mảng nhiều chiều thì có thể truyền vào danh sách các chỉ số.
Reverse(<tên mảng>)	Đảo ngược các giá trị của mảng 1 chiều.
Sort(<tên mảng>)	Sắp xếp các phần tử của mảng 1 chiều.
IndexOf(<tên mảng>, <phần tử cần tìm>)	Tìm kiếm 1 phần tử có tồn tại trong mảng hay không. Nếu có thì trả về vị trí xuất hiện đầu tiên trong mảng ngược lại sẽ trả về 0.

Đây chỉ là một số thuộc tính và phương thức tiêu biểu. Ngoài ra còn nhiều phương thức, thuộc tính khác các bạn có thể tự khám phá.

Lưu ý là các phương thức **Sort**, **Reverse**, **IndexOf** được gọi thông qua tên lớp. Tức là muốn sử dụng các phương thức trên ta sẽ dùng cú pháp:

- **Array.Sort**(<tên mảng>)
- **Array.Reverse**(<tên mảng>)
- **Array.IndexOf**(<tên mảng>, <phần tử cần tìm>)

Nhờ sự hỗ trợ của các phương thức trên mà ta có thể dễ dàng thao tác với dữ liệu mảng như sắp xếp, đảo ngược, ... mà không cần phải viết chi tiết ra giải thuật như thế nào.

Sau đây là một vài ví dụ ứng dụng lớp Array:

**C#:**

```
/*
 * Khai báo và khởi tạo mảng 1 chiều tên IntArray có 4 phần tử chưa được sắp xếp
 */

int[] IntArray = { 5, 2, 1, 3 };

/*
 * Thực hiện câu lệnh sắp xếp mảng
 * Ở đây mặc định là sẽ sắp xếp tăng dần.
 * Nếu bạn muốn sắp xếp giảm dần có thể tận dụng phương thức đảo ngược các giá trị mảng để được mảng giảm dần
 */

Array.Sort(IntArray);
Console.WriteLine(" Mảng đã sắp xếp tăng dần: ");

for (int i = 0; i < IntArray.Length; i++)
{
    Console.Write("\t" + IntArray[i]);
}
Console.WriteLine();

/* Đảo ngược các phần tử của mảng để được 1 mảng giảm dần */
Array.Reverse(IntArray);
Console.WriteLine(" Mảng đã sắp xếp giảm dần: ");

for (int i = 0; i < IntArray.Length; i++)
{
    Console.Write("\t" + IntArray[i]);
}
```

Kết quả khi chạy đoạn chương trình trên:

```
file:///C:/Users/HT/Documents/Visual Studio 2010/Projects/
Mảng đã sắp xếp tăng dần:
1 2 3 5
Mảng đã sắp xếp giảm dần:
5 3 2 1
```

Ngoài ra lớp Array cũng hỗ trợ bạn khai báo sử dụng mảng nhưng thường thì ít ai sử dụng vì nó dài dòng và không tường minh.

## Kết luận

Nội dung bài này giúp các bạn nắm được:

- Khai báo, khởi tạo và sử dụng mảng 3 chiều trong C#.
- Khai báo, khởi tạo và sử dụng mảng jagged trong C#.
- Lớp Array trong C#.

Bài sau chúng ta sẽ tìm hiểu về [CẤU TRÚC LẶP FOREACH 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**”.

