

## I.1. Nạp chồng phương thức

Nạp chồng phương thức (method overloading: nạp chồng hàm) là định nghĩa các hàm cùng tên nhưng khác tham số hoặc kiểu trả về. Khi chạy chương trình, tùy tình huống mà hàm thích hợp nhất được gọi.

### Ví dụ 1:

Minh họa việc nạp chồng phương thức tạo lập để linh động trong cách tạo đối tượng. Lớp *Date* có 3 phương thức tạo lập có tác dụng lần lượt như sau:

- ❖ *public Date()*: khởi tạo đối tượng thuộc lớp *Date* với giá trị mặc định là 1/1/1900.
- ❖ *public Date(int D, int M, int Y)*: khởi tạo các giá trị *Day*, *Month*, *Year* của đối tượng thuộc lớp *Date* bằng ba tham số *D*, *M*, *Y*.
- ❖ *public Date(Date ExistingDate)*: đây là hàm copy constructor, khởi tạo đối tượng mới thuộc lớp *Date* bằng một đối tượng cùng kiểu đã tồn tại.
- ❖ *public Date(System.DateTime dt)*: khởi tạo đối tượng thuộc lớp *Date* bằng dữ liệu của đối tượng thuộc lớp *System.DateTime* (có sẵn).

```
using System;
```

```
public class Date
{
    private int Year;
    private int Month;
    private int Day;

    public void Display( )
    {
        Console.WriteLine("{0}/{1}/{2}", Day, Month, Year);
    }

    // constructors without argument --> set date to 1/1/1900
    public Date()
    {
        Console.WriteLine("Constructor không tham số!");
        Year = 1900;
        Month = 1;
        Day = 1;
    }

    // constructors with DateTime argument
    public Date(System.DateTime dt)
    {
        Console.WriteLine("Constructor với tham số là một đối tượng DateTime!");
        Year = dt.Year;
        Month = dt.Month;
```

```

        Day = dt.Day;
    }
    // constructors with 3 int arguments
    public Date(int D, int M, int Y)
    {
        Console.WriteLine("Constructor co 3 tham so!");
        Year = Y;
        Month = M;
        Day = D;
    }
    // copy constructors
    public Date(Date ExistingDate)
    {
        Console.WriteLine("Copy constructor!");
        Year = ExistingDate.Year;
        Month = ExistingDate.Month;
        Day = ExistingDate.Day;
    }
}

class DateOverLoadConstructorApp
{
    static void Main(string[] args)
    {
        System.DateTime currentTime =
System.DateTime.Now;
        Date t1 = new Date(); // t1 = 1/1/1900
        Console.Write("t1: ");
        t1.Display();
        Console.WriteLine();

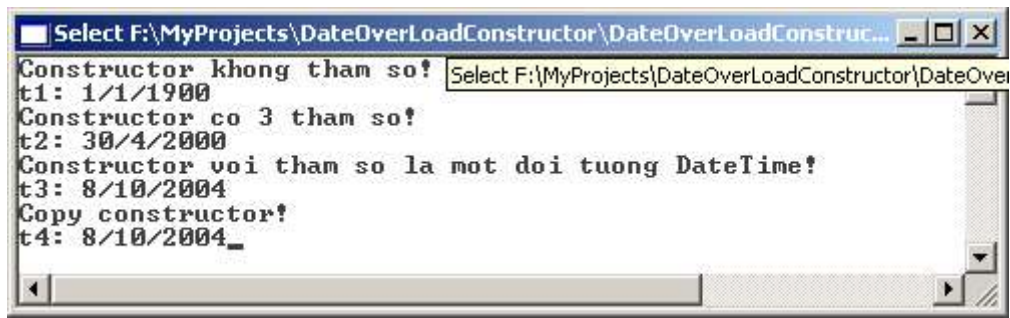
        Date t2 = new Date(30, 4, 2000);
        Console.Write("t2: ");
        t2.Display( );
        Console.WriteLine();

        Date t3 = new Date(currentTime);
        Console.Write("t3: ");
        t3.Display( );
        Console.WriteLine();

        Date t4 = new Date(t3);
        Console.Write("t4: ");
        t4.Display( );
        Console.ReadLine();
    }
}

```

Kết quả của chương trình:



## Ví dụ 2:

Nạp chồng phương thức khởi tạo của lớp phân số để linh động khi tạo ra các đối tượng phân số. (Xem cách trả về của hàm `public PhanSo Cong(PhanSo PS2)`).

```
class PhanSo{
    int Tu, Mau;
    // Hàm khởi tạo gán giá trị cố định
    public PhanSo() {
        Tu = 0;
        Mau = 1;
    }
    public PhanSo(int x) {
        Tu = x;
        Mau = 1;
    }
    public PhanSo(int t, int m) {
        Tu = t;
        Mau = m;
    }

    public void XuatPhanSo() {
        Console.WriteLine("{0}/{1}", Tu, Mau);
    }
    public PhanSo Cong(PhanSo PS2) {
        int TS = Tu * PS2.Mau + Mau * PS2.Tu;
        int MS = Mau * PS2.Mau;
        //Gọi hàm khởi tạo 2 tham số
        PhanSo KetQua = new PhanSo(TS, MS);
        return KetQua;
    }
}

class Program
{
    static void Main(string[] args) {
        p1.XuatPhanSo(); Console.WriteLine();

        PhanSo p2 = new PhanSo(3); // p2 = 3/1
        p2.XuatPhanSo();
        Console.WriteLine();

        Console.WriteLine("Nhập tu so: ");
    }
}
```

```

        int Ts = int.Parse(Console.ReadLine());
        Console.WriteLine("Nhap mau so: ");
        int Ms = int.Parse(Console.ReadLine());
        PhanSo p3 = new PhanSo(Ts, Ms);
        p3.XuatPhanSo();
        Console.WriteLine();

        p1 = p2.Cong(p3);
        p1.XuatPhanSo();
        Console.ReadLine();
    }
}

```

Ta có thể định nghĩa phương thức nạp chồng chỉ khác nhau ở từ khóa *ref* hoặc *out* nhưng không thể có hai phương thức chỉ khác nhau ở hai từ khóa *ref* và *out*.

Chẳng hạn, việc nạp chồng như sau là hợp lệ:

```

class MyClass
{
    public void MyMethod(int i) { i = 10; }
    public void MyMethod(ref int i) { i = 10; }
}

```

nhưng việc nạp chồng như sau là không hợp lệ:

```

class MyClass
{
    public void MyMethod(out int i) { i = 10; }
    public void MyMethod(ref int i) { i = 10; }
}

```

## 1.2. Sử dụng các thành viên tĩnh (static)

Dữ liệu và phương thức của một lớp có thể là thành viên thuộc chung cho tất cả các đối tượng trong lớp đó (còn gọi là thành viên thuộc lớp) hoặc thuộc riêng từng đối tượng.

Thành viên thể hiện (non-static member) được kết hợp riêng với từng đối tượng của lớp (không có từ khóa *static* đứng trước). Khi mỗi đối tượng của lớp được tạo ra, chúng được cấp phát vùng nhớ để chứa các biến thành viên này. Như vậy, trong cùng một lớp, các đối tượng khác nhau có những biến dữ liệu cùng tên, cùng kiểu nhưng được cấp phát ở các vùng nhớ khác nhau và giá trị của chúng cũng có thể khác nhau.

Trong khi đó, thành viên tĩnh (static member) có từ khóa *static* đứng trước. Chúng được coi là phần chung của các đối tượng trong cùng một lớp. Mọi đối tượng thuộc lớp đều có thể truy cập thành viên tĩnh.

Như vậy, các thành viên thể hiện được xem là toàn cục trong phạm vi từng đối tượng còn thành viên tĩnh được xem là toàn cục trong phạm vi một lớp.

Việc truy cập đến thành viên tĩnh phải thực hiện thông qua tên lớp (không được truy cập thành viên tĩnh thông qua đối tượng) theo cú pháp:

***TênLớp.TênThànhViênTĩnh***

### Chú ý:

- Phương thức tĩnh thao tác trên các dữ liệu tĩnh.
- Phương thức tĩnh khi truy cập đến các thành viên không tĩnh phải thông qua một đối tượng, theo cú pháp:

#### **TênĐốiTượng.TênThànhViênTheerHiện**

Ngoài ra, ta có thể định nghĩa một phương thức tạo lập tĩnh. Phương thức này chạy đầu tiên và chỉ một lần khi ta truy cập đến một phần tử nào đó của lớp hoặc đối tượng. Do vậy phương thức này dùng để khởi gán giá trị cho biến tĩnh của lớp và/hoặc thực hiện một số công việc chuẩn bị nào đó.

**Ví dụ:** Biến thành viên tĩnh được dùng với mục đích theo dõi số thể hiện hiện tại của lớp.

```
using System;
public class Cat
{
    private static int SoMeo = - 6; // biến tĩnh
    private string TenMeo ;

    // Phương thức tạo lập của doi tuong
    public Cat( string T) {
        TenMeo = T ;
        Console.WriteLine("WOAW!!!! {0} day!", TenMeo);
        SoMeo++;
    }

    // Phương thức tạo lập tĩnh( không có mức độ truy
    cập public, private...) , được chạy đầu tiên
    static Cat( ) {
        Console.WriteLine("Bat dau lam thit meo !!!!");
        SoMeo = 0;
    }

    public static void HowManyCats( ){
        Console.WriteLine("Dang lam thit {0} con meo!", SoMeo);
    }
}

public class Tester {
    static void Main( ) {
        Cat.HowManyCats( );
        Cat tom = new Cat("Meo Tom" );
        Cat.HowManyCats( );
        Cat muop = new Cat("Meo Muop");
        Cat.HowManyCats( );
        //Tom.HowManyCats( ); ---> Error
        Console.ReadLine();
    }
}
```

}

Trong ví dụ này, ta xây dựng lớp **Cat** với một biến tĩnh **SoMeo** để đếm số thể hiện (số mèo) hiện có và một biến thể hiện **TenMeo** để lưu tên của từng đối tượng mèo. Như vậy, mỗi đối tượng **tom**, **muop** đều có riêng biến **TenMeo** và chúng dùng chung biến **SoMeo**.

Ban đầu biến **SoMeo** được khởi gán giá trị -6, nhưng khi gọi lệnh **Cat.HowManyCats ( )** thì phương thức tạo lập tĩnh **static Cat()** tự động thực hiện trước và gán lại giá trị 0 cho biến tĩnh này. Điều này đảm bảo trước khi tạo ra đối tượng Cat đầu tiên thì biến **SoMeo** luôn = 0.

Mỗi khi một đối tượng thuộc lớp **Cat** được tạo ra thì phương thức tạo lập của đối tượng này truy cập đến biến đếm **SoMeo** và tăng giá trị của biến này lên một đơn vị. Như vậy, khi đối tượng **tom** được tạo ra, giá trị của biến này tăng lên thành 1, khi đối tượng **muop** được tạo ra, giá trị của biến này tăng lên thành 2.

Phương thức tĩnh **HowManyCats ( )** thực hiện nhiệm vụ xuất biến tĩnh **SoMeo** thông qua tên lớp bằng câu lệnh:

```
Cat.HowManyCats ( ) ;
```

Nếu ta gọi lệnh sau thì trình biên dịch sẽ báo lỗi:

```
tom.HowManyCats ( ) ;
```

Kết quả chạy chương trình:



**Bài tập:** Xây dựng lớp **MyDate** lưu trữ các giá trị ngày, tháng, năm với các phương thức: constructor với 3 tham số, xuất, kiểm tra năm nhuận, tính số ngày của tháng theo tháng và năm, xác định ngày kế tiếp của đối tượng ngày/ tháng/ năm hiện hành.

**Gợi ý:** để tính số ngày của tháng ta có thể dùng một mảng tĩnh {31, 28, 31, 30, 31, 30, 31, 31, 30,31, 30,31} để lưu số ngày tương ứng với từng tháng. Tuy nhiên với tháng 2 thì tùy năm có nhuận hay không mà ta tính ra giá trị tương ứng.

### 1.3. Tham số của phương thức

Trong C#, ta có thể truyền tham số cho phương thức theo kiểu tham chiếu hoặc tham trị. Khi truyền theo kiểu tham trị sẽ xảy ra việc sao chép giá trị từ đối số (biến tham số thực khi gọi hàm) sang tham số (biến tham số hình thức khi định nghĩa hàm). Còn khi truyền theo kiểu tham chiếu thì biến đối số và biến tham số đều trỏ tới cùng một vùng nhớ.

Trong C#, từ khóa **ref** để truyền đối số theo kiểu tham chiếu vào trong phương thức. Khi đó biến tham chiếu phải khác **null** trước khi gọi hàm. Biến tham chiếu này nhớ lại thay đổi trong khi gọi hàm.

Từ khóa **out** để truyền đối số cho phương thức theo kiểu tham chiếu mà không cần khởi gán giá trị đầu cho đối số. Biến tham chiếu này được dùng khi hàm có nhiều giá trị trả về và trong phương thức phải có lệnh gán giá trị cho tham chiếu này nếu ban đầu nó bằng **null**.

Đối với những dữ liệu kiểu giá trị (**int**, **long**, **float**, **char**,...), muốn thay đổi giá trị của chúng thông qua việc truyền tham số cho hàm, phương thức ta phải truyền theo kiểu tham chiếu một cách tường minh bằng từ khóa **ref** hoặc **out**.

Đối với những dữ liệu kiểu tham chiếu (đối tượng, mảng), khi dùng chúng để truyền đối số mà không có từ khóa **ref** hoặc **out**, ta chỉ có thể làm thay đổi giá trị của vùng nhớ trong **heap** mà chúng trỏ tới nhưng địa chỉ vùng nhớ mà biến tham chiếu này trỏ tới sẽ không bị thay đổi sau khi kết thúc hàm. Tức là ta chỉ thay đổi được 1 phần dữ liệu của vùng nhớ mà biến tham chiếu này trỏ tới. Việc thay đổi thành vùng nhớ khác không có tác dụng. Nếu muốn thay đổi vùng nhớ mà chúng trỏ tới ta phải dùng từ khóa **ref** hoặc **out** một cách tường minh.

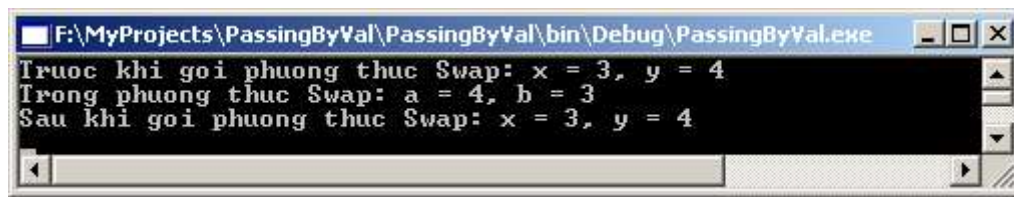
### I.3.1. Truyền tham trị bằng tham số kiểu giá trị

Trong ví dụ sau đây, *a* và *b* là hai tham số dạng tham trị của phương thức *Swap* nên mọi sự thay đổi chỉ diễn ra trong thân phương thức này mà không ảnh hưởng đến đối số *x*, *y* được truyền vào.

```
using System;
class PassingByVal
{
    static void Swap(int a, int b)
    {
        int Temp = a;
        a = b;
        b = Temp;
        Console.WriteLine("Trong phuong thuc Swap: a = {0}, b = {1}", a, b);
    }

    static void Main(string[] args)
    {
        int x = 3, y = 4;
        Console.WriteLine("Truoc khi goi phuong thuc Swap: x = {0}, y = {1}", x, y);
        Swap(x,y);
        Console.WriteLine("Sau khi goi phuong thuc Swap: x = {0}, y = {1}", x, y);
        Console.ReadLine();
    }
}
```

Kết quả chương trình:



```
F:\MyProjects\PassingByVal\PassingByVal\bin\Debug\PassingByVal.exe
Truoc khi gọi phương thức Swap: x = 3, y = 4
Trong phương thức Swap: a = 4, b = 3
Sau khi gọi phương thức Swap: x = 3, y = 4
```

### I.3.2. Truyền tham chiếu bằng tham số kiểu giá trị với từ khóa `ref`

Để phương thức *Swap* cho ra kết quả như mong muốn ta phải sửa lại tham số *a*, *b* theo kiểu tham chiếu như sau:

```
static void Swap(ref int a, ref int b)
```

Khi đó ta gọi phương thức *Swap* với hai đối số *x*, *y* theo cú pháp:

```
Swap(ref x, ref y);
```

Một phương thức chỉ có thể trả về một giá trị, do đó khi muốn phương thức trả về nhiều giá trị, chúng ta dùng cách thức truyền tham chiếu. Ví dụ, phương thức *GetTime* sau đây trả về các giá trị *Hour*, *Minute*, *Second*.

```
using System;
```

```
public class Time {
    private int Hour;
    private int Minute;
    private int Second;

    public void Display( ) {
        Console.WriteLine("{0}:{1}:{2}", Hour, Minute,
Second);
    }

    public Time(System.DateTime dt) {
        Hour = dt.Hour;
        Minute = dt.Minute;
        Second = dt.Second;
    }
    public void GetTime(ref int h, ref int m, ref int
s) {
        h = Hour;
        m = Minute;
        s = Second;
    }
}
```

```
public class PassingParameterByRef {
    static void Main( ) {
        DateTime currentTime = DateTime.Now;
```



```

        Time t = new Time(currentTime);
        t.Display( );

        int theHour = 0;
        int theMinute = 0;
        int theSecond = 0;

        t.GetTime(ref theHour, ref theMinute, ref
theSecond);
        Console.WriteLine("Current time: {0}:{1}:{2}",
theHour, theMinute, theSecond);
        Console.ReadLine();
    }
}

```

### I.3.3. Truyền tham chiếu với tham số kiểu giá trị bằng từ khóa out

Mặc định, C# quy định tất các biến phải được gán giá trị trước khi sử dụng, vì vậy, trong ví dụ trên, nếu chúng ta không khởi tạo các biến *theHour*, *theMinute* bằng giá trị 0 thì trình biên dịch sẽ thông báo lỗi. Từ khóa **out** cho phép ta sử dụng tham chiếu mà không cần phải khởi gán giá trị đầu. Trong ví dụ trên, ta có thể sửa phương thức *GetTime* thành:

```
public void GetTime(out int h, out int m, out int s)
```

và hàm *Main()* được sửa lại như sau:

```

static void Main( )
{
    DateTime currentTime = DateTime.Now;
    Time t = new Time(currentTime);
    t.Display( );
    /*int theHour = 0;
    int theMinute = 0;
    int theSecond = 0;*/
    int theHour;
    int theMinute;
    int theSecond;
    t.GetTime(out theHour, out theMinute, out
theSecond);
    Console.WriteLine("Current time: {0}:{1}:{2}",
theHour, theMinute, theSecond);
    Console.ReadLine();
}

```

### I.3.4. Truyền tham trị với tham số thuộc kiểu tham chiếu

Khi truyền tham số theo cách này ta chỉ có thể thực hiện các thao tác làm thay đổi các dữ liệu thành phần của đối số. Các thao tác làm thay đổi vùng nhớ do đối số trở tới không có tác dụng.

### Ví dụ 1:

Xét hàm *NghichDao2(PhanSo p)* trong ví dụ dưới đây. Phân số *p* là đối tượng nên nó là dữ liệu thuộc kiểu tham chiếu và được truyền cho hàm theo kiểu tham trị, vì vậy, thao tác gán *p = p2* không có tác dụng.

```
class PhanSo{
    int Tu, Mau;

    public PhanSo()    {
        Tu = 0;
        Mau = 1;
    }

    public PhanSo(int x)    {
        Tu = x;
        Mau = 1;
    }

    public PhanSo(int t, int m)    {
        Tu = t;
        Mau = m;
    }

    public void XuatPhanSo()    {
        Console.WriteLine("{0}/{1}", Tu, Mau);
    }
    //Hàm này có tác dụng nghịch đảo vì nó làm thay đổi
    các biến dữ liệu của p nhưng không làm thay đổi vùng
    nhớ của tham trị p.
    public static void NghichDao1(PhanSo p)    {
        int t = p.Tu;
        p.Tu = p.Mau;
        p.Mau = t;
    }
    //Hàm này không có tác dụng nghịch đảo vì nó làm
    thay đổi vùng nhớ của tham trị p.

    public static void NghichDao2(PhanSo p)    {
        PhanSo p2 = new PhanSo();
        p2.Mau = p.Tu;
        p2.Tu = p.Mau;
        p = p2;
    }

    //Hàm này có tác dụng nghịch đảo vì có từ khóa ref
    public static void NghichDao3(ref PhanSo p)    {
        PhanSo p2 = new PhanSo();
        p2.Mau = p.Tu;
        p2.Tu = p.Mau;
```

```

        p = p2;
    }
}
class Program
{
    static void Main(string[] args)
    {
        PhanSo p1 = new PhanSo(3, 1); // p1 = 3/1
        PhanSo p1.XuatPhanSo(); Console.WriteLine();

        PhanSo.NghichDao1(p1);
        p1.XuatPhanSo(); Console.WriteLine();

        PhanSo.NghichDao2(p1);
        p1.XuatPhanSo(); Console.WriteLine();
        PhanSo.NghichDao3(ref p1);
        p1.XuatPhanSo();
        Console.ReadLine();
    }
}

```

## Ví dụ 2:

Ví dụ sau đây minh họa việc truyền tham số là tham chiếu (*myArray*) cho phương thức *Change* theo kiểu tham trị. Đối số *myArray* là một dữ liệu tham chiếu, nó trỏ tới một mảng số nguyên được cấp phát trong **Heap**. Khi truyền *myArray* theo kiểu tham trị cho phương thức *Change*, ta chỉ có thể làm thay đổi các giá trị của vùng nhớ trong **Heap**; tức là, ta chỉ có thể thay đổi giá trị của một ô nhớ trong mảng *myArray*, mọi thao tác cấp phát lại vùng nhớ cho *myArray* thực hiện trong phương thức này sẽ không ảnh hưởng tới mảng gốc được cấp phát trong hàm *Main()*.

```

using System;

class PassingRefByVal
{
    static void Change(int[] arr)
    {
        // Lệnh này thay lam thay doi gia tri cua arr.
        arr[0] = 888;
        // Cap phat lai arr, lam thay doi dia chi cua arr.
        // Lệnh này chỉ có tác dụng cục bộ. Sau khi gọi hàm Change,
        // giá trị của arr[0] vẫn không thay đổi.
        arr = new int[5] { -3, -1, -2, -3, -4 };
        Console.WriteLine("Trong phương thức Change,
            phan tu dau tien la: {0}", arr[0]);
    }

    public static void Main()
    {
        int[] myArray = { 1, 4, 5 };
    }
}

```

```

        Console.WriteLine("Trong ham Main,truoc khi gọi  

        phuong thuc Change, myArray [0] = {0}", myArray  

        [0]);  

        Change(myArray);  

        Console.WriteLine("Trong ham Main, sau khi gọi  

        phuong thuc Change, myArray [0] = {0}", myArray  

        [0]);  

        Console.ReadLine();  

    }  

}

```

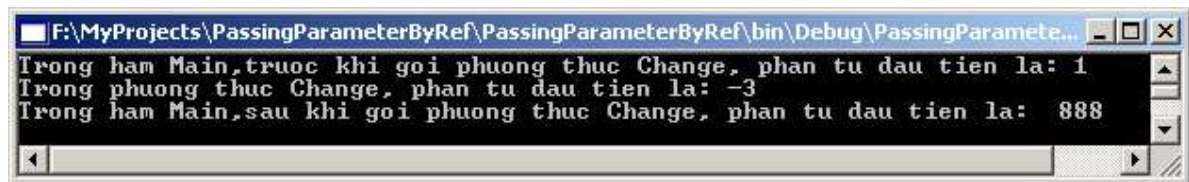
Kết quả chương trình cho thấy: trong hàm **Change()**, ta chỉ có thể thay đổi giá trị của vùng nhớ do *myArray* trỏ tới (được cấp phát trong **heap**) bằng lệnh:

```
arr[0] = 888;
```

mà không thay đổi được bản thân tham chiếu *myArray* bằng lệnh cấp phát lại:

```
arr = new int[5] {-3, -1, -2, -3, -4};
```

Vì nếu thay đổi được tham chiếu *myArray* thì sau khi gọi phương thức *Change* giá trị của phần tử đầu tiên trong mảng phải bằng -3.



### I.3.5. Truyền tham chiếu với tham số thuộc kiểu dữ liệu tham chiếu

Với cách truyền tham số này, những câu lệnh làm thay đổi tham số sẽ có hiệu lực.

Trong ví dụ 2 phần II.7.4 nếu ta khai báo lại nguyên mẫu của phương thức *Change* như sau:

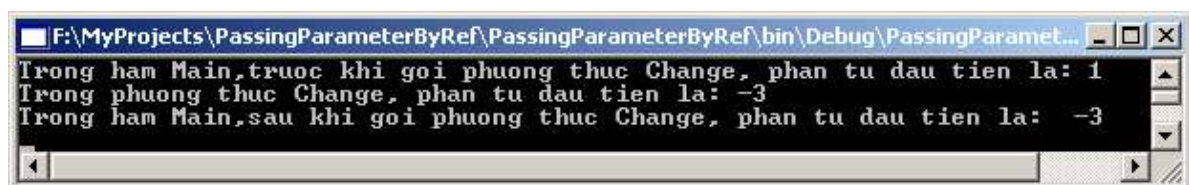
```
static void Change(ref int[] arr)
```

và trong hàm *Main()* ta thay câu lệnh **Change(myArray)** ; bằng câu lệnh:

```
Change(ref myArray);
```

thì mọi thao tác làm thay đổi giá trị của vùng nhớ trong **heap** do *myArray* trỏ tới hoặc mọi thao tác cấp phát lại vùng nhớ cho *myArray* thực hiện trong phương thức này sẽ làm thay đổi mảng gốc được cấp phát trong hàm *Main()*.

Kết quả chạy chương trình:



Điều này chứng tỏ ta có thể thay đổi đối số thuộc kiểu tham chiếu.

Ta có thể lý giải tương tự cho hàm **NghichDao3** ([ref PhanSo p](#)) trong ví dụ 1 phần II.7.4.

**Bài tập 1:** Hãy viết phương thức để hoán vị hai dữ liệu kiểu chuỗi (string). Hãy giải thích tại sao khi dùng phương thức này ta cần phải truyền tham số theo kiểu tham chiếu cho phương thức dù string là dữ liệu thuộc kiểu tham chiếu.

**Bài tập 2:** Viết chương trình nhập một lớp gồm  $N$  học sinh, mỗi học sinh có các thông tin như: họ, tên, điểm văn, điểm toán, điểm trung bình. Tính điểm trung bình của từng học sinh theo công thức:  $(\text{điểm văn} + \text{điểm toán})/2$ .

- Tính trung bình điểm văn của cả lớp.
- Tính trung bình điểm toán của cả lớp.
- Sắp xếp học sinh trong lớp theo thứ tự họ tên.
- Sắp xếp học sinh trong lớp theo thứ tự không giảm của điểm trung bình, nếu điểm trung bình bằng nhau thì sắp xếp theo tên.

#### 1.4. Tham chiếu this

Trong lập trình đơn tuyến với C#, tại mỗi thời điểm chỉ có một hàm được thực thi. Mỗi khi gọi thực thi một phương thức của một đối tượng (không phải là phương thức tĩnh) thì tham chiếu **this** tự động trỏ đến đối tượng này. Mọi phương thức của đối tượng đều có thể tham chiếu đến các thành phần của đối tượng thông qua tham chiếu **this**. Có 3 trường hợp thường dùng tham chiếu **this**:

- Tránh xung đột tên khi tham số của phương thức trùng tên với tên biến dữ liệu của đối tượng.
- Dùng để truyền đối tượng hiện tại làm tham số cho một phương thức khác (chẳng hạn gọi đệ qui)
- Dùng với mục đích chỉ mục.

**Ví dụ 1:** Dùng tham chiếu **this** với mục đích tránh xung đột tên của tham số với tên biến dữ liệu của đối tượng.

```
public class Date
{
    private int Year;
    private int Month;
    private int Day;

    public Date(int Day, int Month, int Year)
    {
        Console.WriteLine("Constructor có 3 tham số!");
        this.Year = Year;
        this.Month = Month;
        this.Day = Day;
    }
    ... các phương thức khác
}
```

Ví dụ 2: bài toán tháp Hà Nội, minh họa dùng tham chiếu *this* trong đệ qui.

```
using System;

class Cot
{
    uint [] Disks;
    uint Size, Top = 0;
    char Ten;

    public Cot(uint n, char TenCot)    {
        Size = n;
        Ten = TenCot;
        Disks = new uint[Size];
    }

    public void ThemDia(uint DiskID)    {
        Disks[Top] = DiskID;
        Top++;
    }

    // ~Cot()    {Console.WriteLine("Freeing {0}", Ten);}

    // Chuyen mot dia tren dinh tu cot hien hanh sang cot C
    // Ham nay su dung tham chieu this de chuong trinh duoc
    // ro rang hon
    public void Chuyen1Dia(Cot C) {
        this.Top--;
        Console.WriteLine("Chuyen dia {0} tu {1} sang
        {2}",    this.Disks[Top], this.Ten, C.Ten);

        //C.ThemDia(this.Disks[this.Top]);
        C.Disks[C.Top] = this.Disks[this.Top];
        C.Top++;
    }

    // Chuyen N dia (SoDia) tu cot hien hanh sang cot C, lay
    // cot B lam trung //gian. Ham nay su dung tham chieu this
    // voi muc dich de qui
    public void ChuyenNhieuDia(uint SoDia, Cot B, Cot
    C) {
        // Chuyen mot dia sang C
        if( SoDia == 1) Chuyen1Dia(C);
        else    {
            ChuyenNhieuDia(SoDia-1,C, B);
            Chuyen1Dia(C);
            B.ChuyenNhieuDia(SoDia -1, this, C);
        }
    }
}

class This_ThapHaNoiApp
```

```

{
    static void Main() {
        Cot A, B, C;
        uint SoDia = 4;
        A = new Cot(SoDia, 'A');
        B = new Cot(SoDia, 'B');
        C = new Cot(SoDia, 'C');

        //Nap N dia vao cot A
        uint i = SoDia;
        while(i>0) {
            A. ThemDia(i);
            i--;
        }

        //Chuyen N dia tu cot A sang cot C, lay cot B lam trung
        gian
        A.ChuyenNhieuDia(SoDia,B, C);
        Console.ReadLine();
    }
}

```