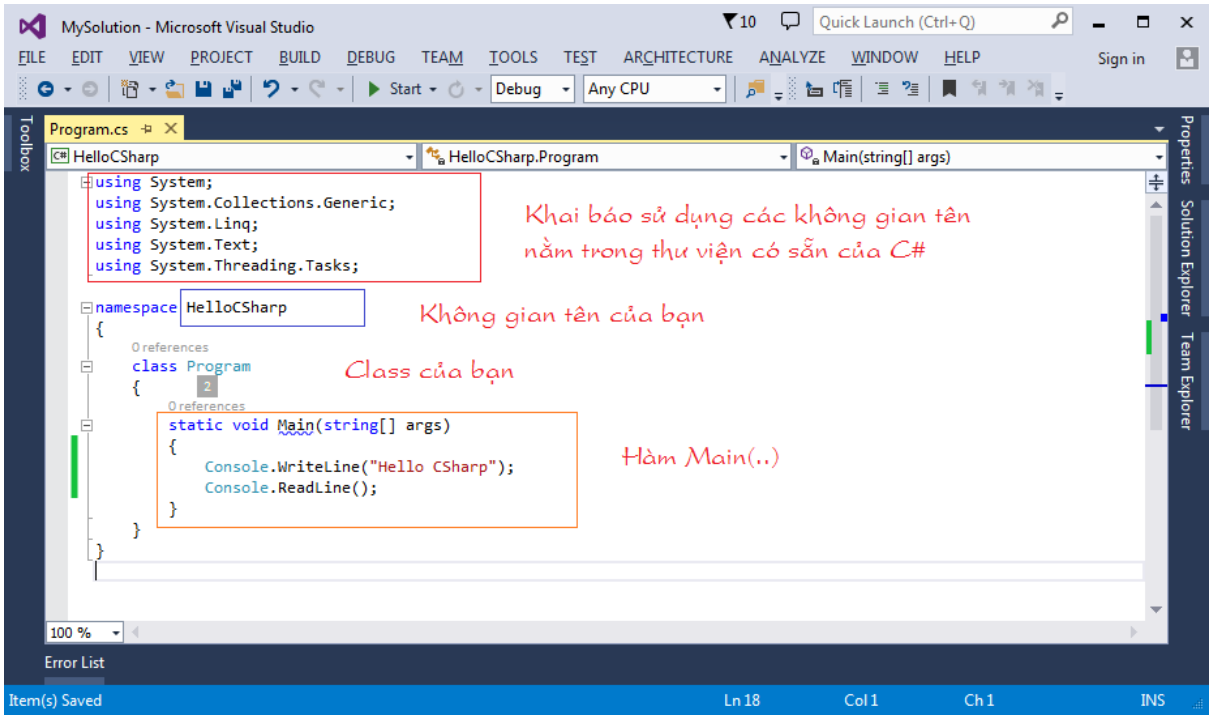
C# Note

- Một không gian tên (namespace) có thể chứa 1 hoặc nhiều class.



- Nếu muốn sử dụng một lớp nào đó, thì phải khai báo sử dụng lớp đó, hoặc khai báo sử dụng không gian tên chứa lớp đó.

```
// Khai báo sử dụng namespace System.
// (Nghĩa là có thể sử dụng tất cả các class có trong namespace này).
using System;
```

- **static** là từ khóa thông báo rằng đây là phương thức tĩnh.
- **void** là từ khóa thông báo rằng phương thức này không trả về gì cả.
- **args** là tham số của phương thức nó là một mảng các **string** (chuỗi) - **string[]**.

```
static void Main(string[] args)
{
        // Ghi ra màn hình Console một dòng chữ.
        Console.WriteLine("Hello CSharp");

        // Đợi người dùng gõ vào một dòng chữ trước khi tắt màn hình Console.(phải nhấn enter để thoát).
        Console.ReadLine();
}
// Khai báo sử dụng không gian tên System
// (Nó có chứa class Console).
using System;

// Và có thể sử dụng class Console:
Console.WriteLine("Hello CSharp");
```

-
- `// Nếu bạn không muốn khai báo sử dụng không gian tên`
- `// Nhưng muốn in ra một dòng text, bạn phải viết đầy đủ:`
- `System.Console.WriteLine("Hello CSharp");`
- Trong một ứng dụng **C#** cần khai báo rõ ràng một lớp có phương thức **Main(string[])** dùng làm điểm bắt đầu để chạy một ứng dụng.

- Các kiểu dữ liệu trong C#

Kiểu	Mô tả	Phạm vi	Giá trị mặc định
bool	Giá trị Boolean (Đúng hoặc sai).	True hoặc False	False
byte	Số tự nhiên không dấu 8-bit	0 tới 255	0
char	Ký tự unicode 16-bit	U +0000 tới U +ffff	'\0'
decimal	Có độ chính xác đến 28 con số và giá trị thập phân (Sử dụng 128-bit)	(-7.9 x 1028 tới 7.9 x 1028) / 100 tới 28	0.0M
double	Kiểu dấu chấm động có độ chính xác gấp đôi (Sử dụng 64-bit)	(+/-)5.0 x 10-324 tới (+/-)1.7 x 10308	0.0D
float	Kiểu dấu chấm động (Sử dụng 32-bit)	-3.4 x 1038 to + 3.4 x 1038	0.0F
int	Số nguyên có dấu 32-bit	-2,147,483,648 tới 2,147,483,647	0
long	64-bit signed integer type	-923,372,036,854,775,808 tới 9,223,372,036,854,775,807	0L
sbyte	Số nguyên có dấu 8-bit	-128 tới 127	0
short	Số nguyên có dấu 16-bit	-32,768 tới 32,767	0
uint	Số nguyên không dấu 32-bit	0 tới 4,294,967,295	0
ulong	Số nguyên không dấu 64-bit	0 tới 18,446,744,073,709,551,615	0
ushort	Số nguyên không dấu 16-bit	0 tới 65,535	0

Biến và khai báo

- Mỗi biến trong C# có một kiểu dữ liệu cụ thể.Trong đó xác định kích thước và phạm vi giá trị có thể được lưu trữ trong bộ nhớ, và tập hợp các toán tử có thể áp dụng cho biến
- Biến có thể thay đổi giá trị trong quá trình tồn tại của nó trong chương trình.
- Các biến có giá trị cố định được gọi là các hằng số, sử dụng từ kháo const để khai báo một biến là hằng số.

```
// Khai báo một biến.
<kiểu dữ liệu> <tên biến>;

// Khai báo một biến đồng thời gián luôn giá trị.
<kiểu dữ liệu> <tên biến> = <giá trị>;

// Khai báo một hằng số:
const <kiểu dữ liệu> <tên hằng số> = <giá trị>;
```

Câu lệnh rẽ nhánh

Câu lệnh If-else

if là một câu lệnh kiểm tra một điều kiện gì đó trong **C#**. Chẳng hạn: Nếu **a > b** thì làm gì đó ....
Các toán tử so sánh thông dụng:

Toán tử	**Ý nghĩa**	**Ví dụ**
>	Lớn hơn	5 > 4 là đúng (true)
<	Nhỏ hơn	4 < 5 là đúng (true)
>=	Lớn hơn hoặc bằng	4 >= 4 là đúng (true)
<=	Nhỏ hơn hoặc bằng	3 <= 4 là đúng (true)
==	Bằng nhau	1 == 1 là đúng (true)
!=	Không bằng nhau	1 != 2 là đúng (true)
&&	Và	a > 4 && a < 10

||	Hoặc	a == 1 || a == 4

```
// Cú pháp

if ( điều kiện)
{
   // Làm gì đó tại đây.
}
// Ví dụ 1:
if ( 5 < 10 )
{
  Console.WriteLine( "Five is now less than ten");
}

// Ví dụ 2:
if ( true )
{
  Console.WriteLine( "Do something here");
}
```

Cấu trúc đầy đủ của **if - else if - else**:

```
// Chú ý rằng sẽ chỉ có nhiều nhất một khối được chạy
// Chương trình kiểm tra điều kiện từ trên xuống dưới khi bắt gặp một điều
// kiện đúng khối lệnh tại đó sẽ được chạy, và chương trình không kiểm tra
tiếp các điều kiện
// còn lại trong cấu trúc rẽ nhánh.

// Nếu điều kiện 1 đúng thì  ...
if (điều kiện 1)
{
  // ... làm gì đó khi điều kiện một đúng.
}
// Ngược lại nếu điều kiện 2 đúng thì ....
else if(điều kiện 2 )
{
// ... làm gì đó khi điều kiện 2 đúng (điều kiện 1 sai).
}
// Ngược lại nếu điều kiện N đúng thì ...
else if(điều kiện N)
{
  // .. làm gì đó khi điều kiện N đúng (các điều kiện ở trên sai)
}
// Ngược lại nếu các điều kiện ở trên đều sai thì
else
{
  // ... làm gì đó tại đây khi mọi điều kiện trên đều sai.
}
```

- **Kiểm tra chuỗi nhập vào có phải là số hay không?**

```
int age;
        string inputAge = Console.ReadLine();
        if (Int32.TryParse(inputAge, out age)){

}else{ // chuoi nhap vao phai la so}
```

Câu lệnh Switch-Case

Cú pháp câu lệnh rẽ nhánh ***switch***:

```
// Sử dụng switch để kiểm tra một giá trị của một biến
switch ( <variable> )
{
case value1:
// Làm gì đó nếu giá trị của biến == value1
break;
case value2:
// Làm gì đó nếu giá trị của biến == value2
break;
...
default:
// Làm điều gì đó tại đây nếu giá trị của biến không thuộc các giá trị liệt kê ở trên.
break;
}
```

*Lệnh **break** trong trường hợp này nói với chương trình rằng thoát ra khỏi **switch**.*

Ví dụ:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class BreakExample
  {
      static void Main(string[] args)
      {
          // Đề nghị người dùng chọn 1 lựa chọn.
          Console.WriteLine("Please select one option:\n");

          Console.WriteLine("1 - Play a game \n");
          Console.WriteLine("2 - Play music \n");
          Console.WriteLine("3 - Shutdown computer \n");
```

```
            // Khai báo một biến option
            int option;

            // Chuỗi người dùng nhập vào từ bàn phím
            string inputStr = Console.ReadLine();

            // Chuyển chuỗi thành số nguyên.
            option = Int32.Parse(inputStr);

            // Kiểm tra giá trị của option
            switch (option)
            {

                case 1:
                    Console.WriteLine("You choose to play the game");
                    break;
                case 2:
                    Console.WriteLine("You choose to play the music");
                    break;
                case 3:
                    Console.WriteLine("You choose to shutdown the computer");
                    break;
                default:
                    Console.WriteLine("Nothing to do...");
                    break;
            }

            Console.ReadLine();
        }
    }
}
```

Bạn có thể gộp nhiều trường hợp (**case**) để thực thi cùng một khối lệnh.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class BreakExample2
  {
      static void Main(string[] args)
      {
          // Khai báo biến option và gán giá trị 3.
          int option = 3;
```

```
            Console.WriteLine("Option = {0}", option);

            // Kiểm tra giá trị của option
            switch (option)
            {

                case 1:
                    Console.WriteLine("Case 1");
                    break;
                // Trường hợp chọn 2,3,4,5 sử lý giống nhau.
                case 2:
                case 3:
                case 4:
                case 5:
                    Console.WriteLine("Case 2,3,4,5!!!");
                    break;
                default:
                    Console.WriteLine("Nothing to do...");
                    break;
            }

            Console.ReadLine();
        }
    }
}
```

Vòng lặp trong C#

Vòng lặp (loop) được sử dụng để chạy lặp lại một khối lệnh. Nó làm chương trình của bạn thực thi lặp đi lặp lại một khối lệnh nhiều lần, đây là một trong các nhiệm vụ cơ bản trong lập trình.

C# hỗ trợ 3 loại vòng lặp khác nhau:

- **FOR**
- **WHILE**
- **DO WHILE**

Vòng lặp for

Cấu trúc của vòng lặp **for**:

```
for ( khởi tạo biến; điều kiện; cập nhập giá trị mới cho biến )
{
   // Thực thi khối lệnh khi điều kiện còn đúng
}
// Ví dụ 1:
// Tạo một biến x và gán giá trị ban đầu của nó là 0
```

```
// Điều kiện kiểm tra là x < 5
// Nếu x < 5 đúng thì khối lệnh được chạy
// Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là
tăng x lên 1.
for (int  x = 0;  x < 5 ; x = x + 1)
{
  // Làm gì đó tại đây khi x < 5 đúng.
}


// Ví dụ 2:
// Tạo một biến x và gán giá trị ban đầu của nó là 2
// Điều kiện kiểm tra là x < 15
// Nếu x < 15 đúng thì khối lệnh được chạy
// Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là
tăng x lên 3.
for (int  x = 2;  x < 15 ; x = x + 3)
{
  // Làm gì đó tại đây khi x < 15 đúng.
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class ForLoopExample
  {
      static void Main(string[] args)
      {
          Console.WriteLine("For loop example");

          // Tạo một biến x và gán giá trị ban đầu của nó là 2
          // Điều kiện kiểm tra là x < 15
          // Nếu x < 15 đúng thì khối lệnh được chạy
          // Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới
tại đây là tăng x lên 3.
          for (int x = 2; x < 15; x = x + 3)
          {
              Console.WriteLine( );
              Console.WriteLine("Value of x = {0}", x);
          }

          Console.ReadLine();
      }
  }
```

```
}
```

Vòng lặp while

Cú pháp của vòng lặp **while**:

```
while (điều kiên)
{
   // Trong khi điều kiện đúng thì thực thi khối lệnh.
}
```

Ví dụ:

```
// Khai báo một biến x
int x = 2;

while ( x < 10)
{
   // Làm gì đó tại đây khi x < 10 còn đúng.
   // ....
   // Cập nhập giá trị mới cho biến x.
   x = x + 3;
}
Ví dụ:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class WhileLoopExample
  {
      static void Main(string[] args)
      {

          Console.WriteLine("While loop example");


          // Tạo một biến x và gán giá trị ban đầu của nó là 2
          int x = 2;

          // Điều kiện kiểm tra là x < 10
          // Nếu x < 10 đúng thì khối lệnh được chạy.
          while (x < 10)
          {
              Console.WriteLine("Value of x = {0}", x);

              x = x + 3;
          }
```

```
            Console.ReadLine();
        }
    }
}
```

Vòng lặp do-while

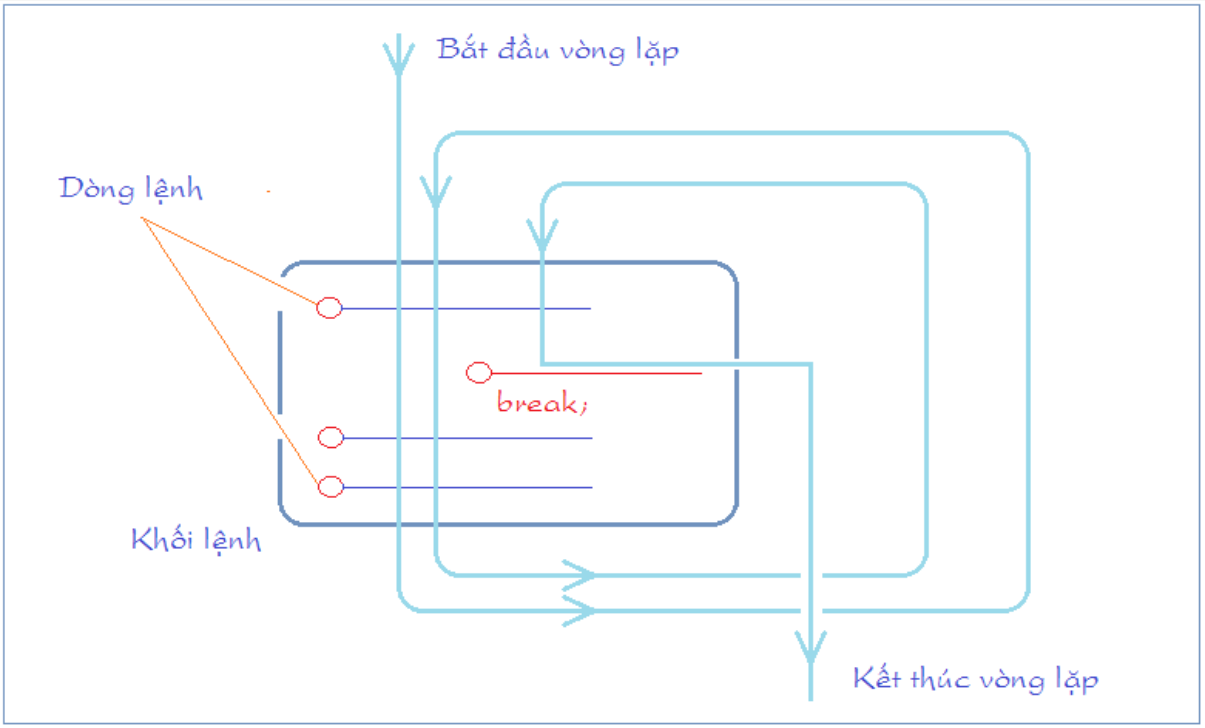
Cú pháp của vòng lặp **do-while:**

```
// Đặc điểm của vòng lặp DO-WHILE là nó sẽ thực khi khối lệnh ít nhất 1 lần.
// Mỗi lần chạy xong khối lệnh nó lại kiểm tra điều kiện xem có thực thi tiếp không.
do
{
   // Làm gì đó tại đây
   // Sau đó mới kiểm tra tiếp điều kiện xem có tiếp tục chạy khối lệnh này nữa hay không.
} while ( điều kiện);  // Chú ý: cần có dấu chấm phẩy tại đây.
Ví dụ:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class DoWhileLoopExample
  {
      static void Main(string[] args)
      {

          Console.WriteLine("Do-While loop example");

          // Tạo một biến x và gán giá trị ban đầu của nó là 2
          int x = 2;

          // Thực hiện khối lệnh.
          // Sau đó kiểm tra điều kiện xem có thực hiện khối lệnh nữa không.
          do
          {
              Console.WriteLine("Value of x = {0}", x);

              x = x + 3;

          } while (x < 10); // Chú ý: Cần có dấu chấm phẩy tại đây.
```

```
            Console.ReadLine();
        }
    }
}
```

Lệnh break trong vòng lặp

break là một lệnh nó có thể nằm trong một khối lệnh của một vòng lặp. Đây là lệnh kết thúc vòng lặp vô điều kiện.

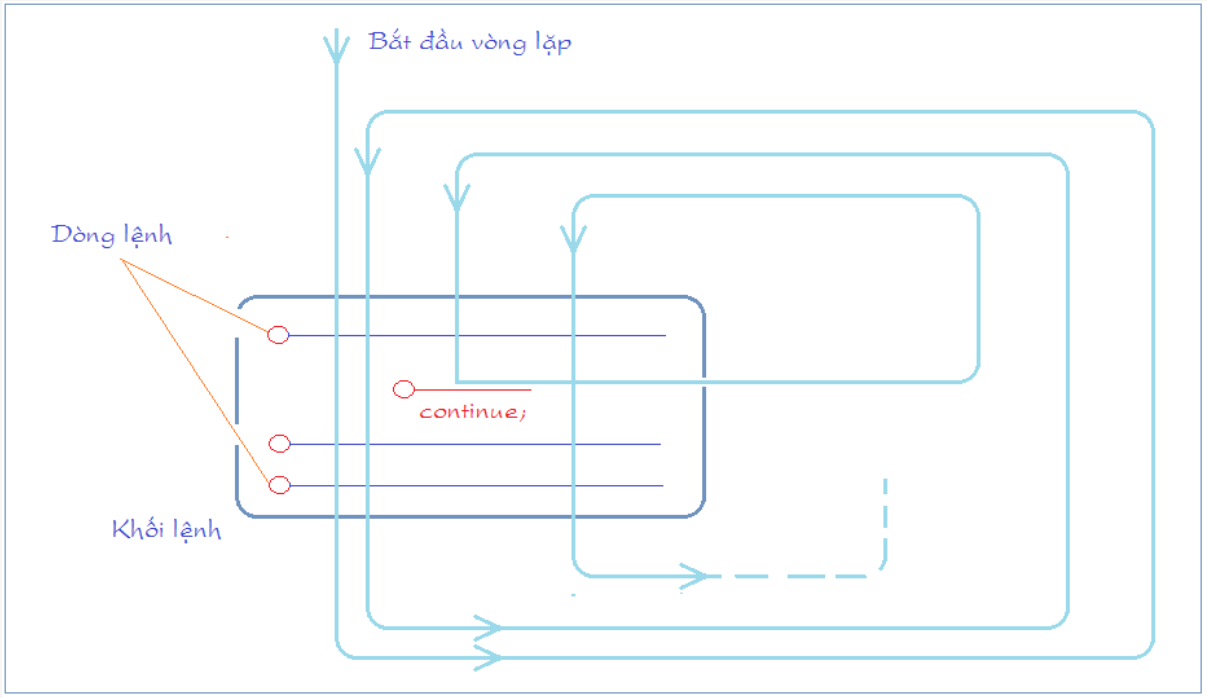



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class LoopBreakExample
  {
      static void Main(string[] args)
      {
          Console.WriteLine("Break example");



          // Tạo một biến x và gán giá trị ban đầu của nó là 2
          int x = 2;
```

```
            while (x < 15)
            {

                Console.WriteLine("----------------------\n");
                Console.WriteLine("x = {0}", x);

                // Kiểm tra nếu x = 5 thì thoát ra khỏi vòng lặp.
                if (x == 5)
                {
                    break;
                }
                // Tăng  x lên 1 (Viết ngắn gọn cho x = x + 1;).
                x++;
                Console.WriteLine("x after ++ = {0}", x);

            }

            Console.ReadLine();
        }
    }
}
```

Lệnh continue trong vòng lặp

continue **là một lệnh, nó có thể nằm trong một vòng lặp, khi bắt gặp lệnh continue chương trình sẽ bỏ qua các dòng lệnh trong khối phía dưới của *continue* và bắt đầu một vòng lặp mới.**

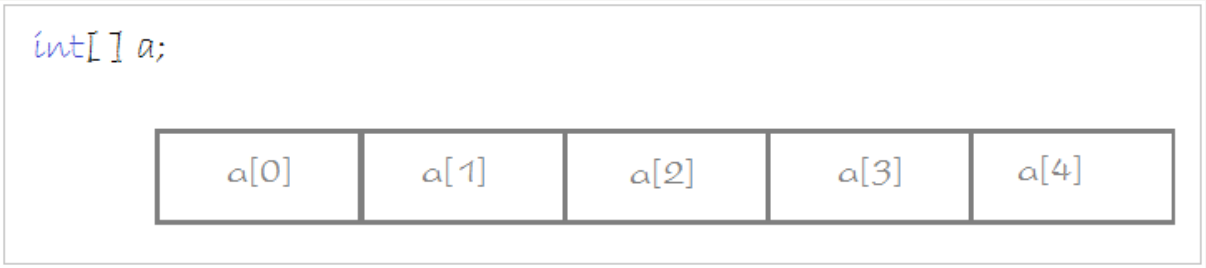



```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class LoopContinueExample
  {
      static void Main(string[] args)
      {

          Console.WriteLine("Continue example");


          // Tạo một biến x và gán giá trị ban đầu của nó là 2
          int x = 2;

          while (x < 7)
          {

              Console.WriteLine("----------------------\n");
              Console.WriteLine("x = {0}", x);

              // % là phép chia lấy số dư
              // Nếu x chẵn, thì bỏ qua các dòng lệnh phía dưới
              // của continue, tiếp tục vòng lặp mới (nếu có).
              if (x % 2 == 0)
              {
                  // Tăng  x lên 1 (Viết ngắn gọn cho x = x + 1;).
                  x++;
                  continue;
              }
              else
              {
                  // Tăng  x lên 1 (Viết ngắn gọn cho x = x + 1;).
                  x++;
              }
              Console.WriteLine("x after ++ = {0}", x);

          }

          Console.ReadLine();
      }
  }
}
```

Mảng trong C#

Mảng một chiều

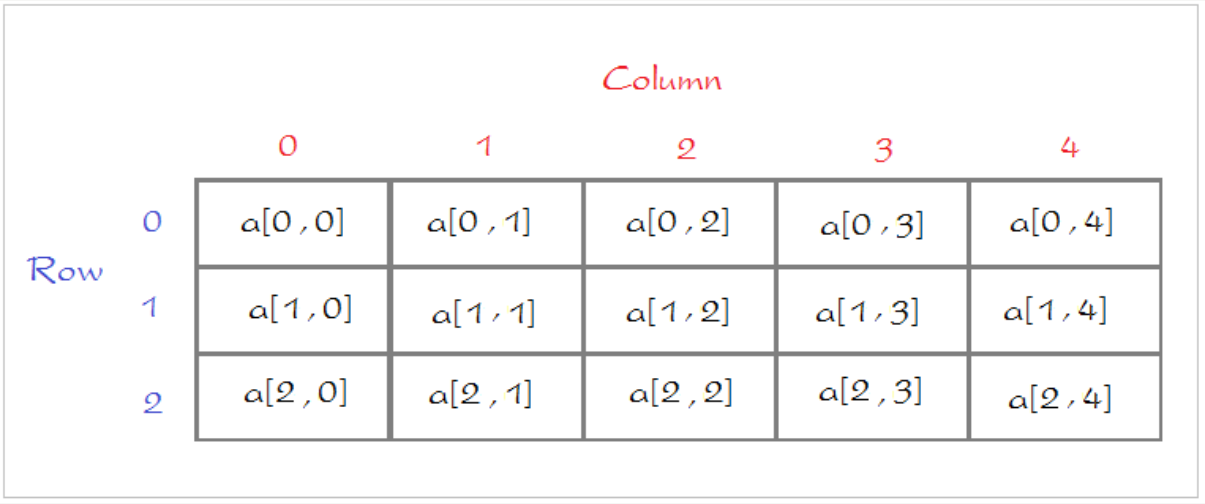


Cú pháp khai báo mảng một chiều:

```
// Cách 1:
// Khai báo một mảng các số int, chỉ rõ các phần tử.
int[] years = { 2001, 2003, 2005, 1980, 2003 };

// Cách 2:
// Khai báo một mảng các số float, chỉ rõ số phần tử.
// (3 phần tử).
float[] salaries = new float[3];
Ví dụ:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class ArrayExample1
  {
      static void Main(string[] args)
      {

          // Cách 1:
          // Khai báo một mảng, gán luôn các giá trị.
          int[] years = { 2001, 2003, 2005, 1980, 2003 };

          // Length là một thuộc tính của mảng, nó trả về số phần tử của
mảng.
          Console.WriteLine("Element count of array years = {0} \n",
years.Length);

          // Sử dụng vòng lặp for để in ra các phần tử của mảng.
          for (int i = 0; i < years.Length; i++) {
              Console.WriteLine("Element at {0} = {1}", i, years[i]);
          }
```

```
            // Cách 2:
            // Khai báo một mảng có 3 phần tử.
            float[] salaries = new float[3];

            // Gán các giá trị cho các phần tử.
            salaries[0] = 1000;
            salaries[1] = 1200;
            salaries[2] = 1100;

            Console.ReadLine();
        }
    }
}
```

Mảng hai chiều



Cú pháp khai báo một mảng 2 chiều:

```
// Khai báo mảng 2 chiều chỉ định các phần tử.
// 3 hàng & 5 cột
int[,] a =   new int[,]  {
    {1 , 2 , 3,  4,   5} ,
    {0,  3,  4,  5,   7},
    {0,  3,  4,  0,   0}
};

// Khai báo một mảng 2 chiều, số dòng 3, số cột 5.
// Các phần tử chưa được gán giá trị.
int[,] a = new int[3,5];
Ví dụ:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace HelloCSharp
{
  class ArrayExample2
  {
      static void Main(string[] args)
      {

          // Khai báo một mảng 2 chiều
          // Khởi tạo sẵn các giá trị.
          int[,] a =   {
                        { 1, 2, 3, 4, 5 },
                        { 0, 3, 4, 5, 7 },
                        { 0, 3, 4, 0, 0 }
                        };

          // Sử dụng vòng lặp for để in ra các phần tử của mảng.
          for (int row = 0; row < 3; row++) {
              for (int col = 0; col < 5; col++) {
                  Console.WriteLine("Element at [{0},{1}] = {2}", row, col,
a[row,col]);
              }
              Console.WriteLine("-------------");
          }

          // Khai báo một mảng 2 chiều có số dòng 3, số cột 5
          // Các phần tử chưa được gán giá trị.
          int[,] b = new int[3, 5];

          Console.ReadLine();
      }
  }
}
```

Mảng của mảng

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class ArrayOfArrayExample
  {
      static void Main(string[] args)
      {
```

```
            // Khai báo một mảng 3 phần tử.
            // Mỗi phần tử là một mảng khác.
            string[][] teams = new string[3][];

            string[] mu = { "Beckham","Giggs"};
            string[] asenal = { "Oezil", "Szczęsny", "Walcott" };
            string[] chelsea = {"Oscar","Hazard","Drogba" };

            teams[0] = mu;
            teams[1] = asenal;
            teams[2] = chelsea;

            // Sử dụng vòng lặp for để in ra các phần tử của mảng.
            for (int row = 0; row < teams.Length; row++)
            {
                for (int col = 0; col < teams[row].Length ; col++)
                {
                    Console.WriteLine("Element at [{0}],[{1}] = {2}", row,
col, teams[row][col]);
                }
                Console.WriteLine("-------------");
            }


            Console.ReadLine();
        }
    }
}
```

Class, đối tượng và cấu tử

- Class
- Cấu tử (Constructor)

Constructors (được gọi là hàm tạo) là những hàm đặc biệt cho phép thực thi, điều khiển chương trình ngay khi khởi tạo đối tượng. Trong C#, Constructors có tên giống như tên của Class và không có giá trị trả về.

- Đối tượng (Instance)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class Person
  {
      // Đây là một trường (Field)
```

```
        // Lưu trữ tên người.
        public string Name;

        // Đây là một cấu tử, còn gọi là phương thức khởi tạo (Constructor)
        // Dùng nó để khởi tạo đối tượng.
        // Cấu tử này có một tham số.
        // Cấu tử luôn có tên giống tên class.
        public Person(string persionName)
        {
            // Gán giá trị từ tham số vào cho trường name.
            this.Name = persionName;
        }

        // Đây là một phương thức trả về kiểu string.
        public string GetName()
        {
            return this.Name;
        }
    }
}
```

Như trên class Person không có phương thức Main. Tiếp theo class PersonTest là ví dụ khởi tạo các đối tượng của Person thông qua các cấu tử.

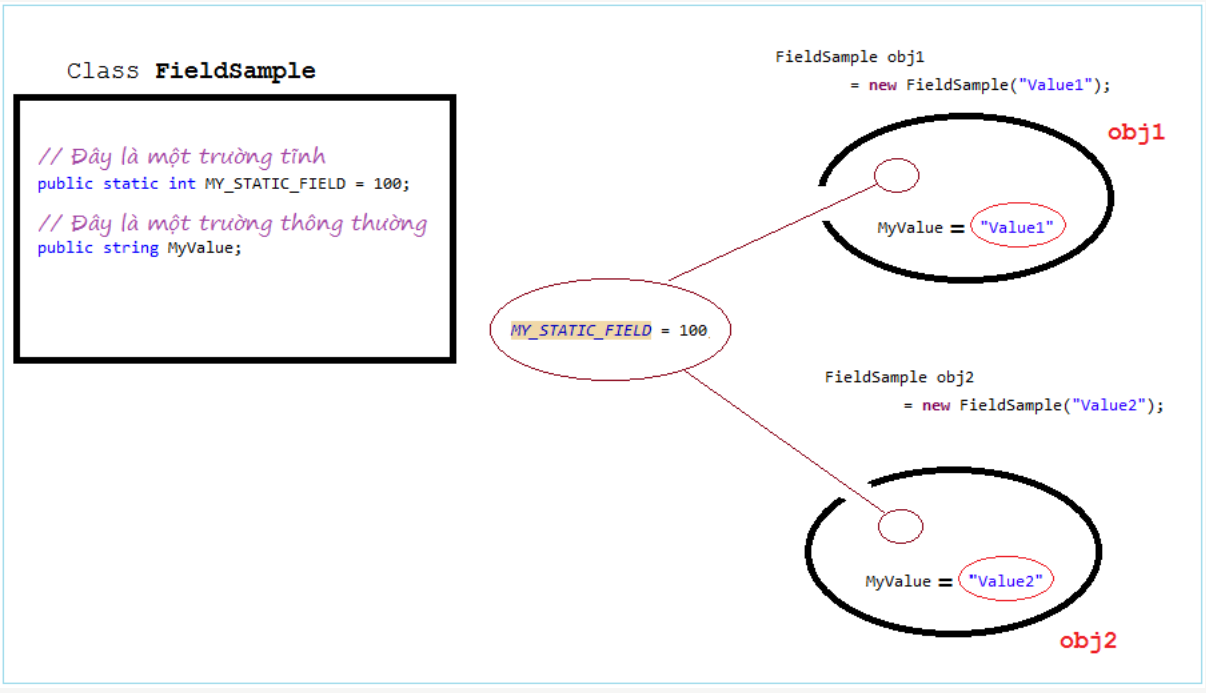
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace HelloCSharp
{
    class PersonTest
    {
        static void Main(string[] args)
        {

            // Tạo một đối tượng từ class Person
            // Khởi tạo đối tượng này tử cấu tử của class Person
            // Cụ thể là Edison
            Person edison = new Person("Edison");

            // Class Person có hàm getName()
            // Sử dụng đối tượng để gọi hàm getName():
            String name = edison.GetName();
            Console.WriteLine("Person 1: " + name);

            // Tạo một đối tượng từ class Person.
            // Khởi tạo đối tượng này tử cấu tử của class Person
```

```
            // Cụ thể là Bill Gates
            Person billGate = new Person("Bill Gates");

            // Class Person có trường name (public)
            // Sử dụng đối tượng để tham chiếu tới nó.
            String name2 = billGate.Name;
            Console.WriteLine("Person 2: " + name2);

            Console.ReadLine();
        }
    }
}
```

Trường (Field)

Trong phần tiếp theo này chúng ta sẽ thảo luận về một số khái niệm:

- **Trường (Field)**
  - **Trường thông thường**
  - **Trường tĩnh (static Field)**
  - **Trường const (const Field)**
  - **Trường tĩnh và readonly (static readonly Field)**




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
```

```
{
  class FieldSample
  {
      // Đây là một trường tĩnh.
      public static int MY_STATIC_FIELD = 100;

      // Đây là một trường thông thường.
      public string MyValue;


      // Cấu tử khởi tạo đối tượng FieldSample.
      public FieldSample(string value)
      {
          this.MyValue = value;
      }
  }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class FieldSampleTest
  {
      static void Main(string[] args)
      {
          // In ra giá trị của trường static.
          // Với các trường tĩnh, bạn phải  truy cập tới nó thông qua
class.
          Console.WriteLine("FieldSample.MY_STATIC_FIELD= {0}",
FieldSample.MY_STATIC_FIELD);

          // Bạn có thể thay đổi giá trị của trường tĩnh.
          FieldSample.MY_STATIC_FIELD = 200;

          Console.WriteLine(" ------------- ");

          // Tạo đối tượng thứ nhất.
          FieldSample obj1 = new FieldSample("Value1");

          // Các trường không tĩnh bạn phải truy cập thông qua đối tượng.
          Console.WriteLine("obj1.MyValue= {0}",  obj1.MyValue);


          // Tạo đối tượng thứ 2:
```

```
            FieldSample obj2 = new FieldSample("Value2");

            //
            Console.WriteLine("obj2.MyValue= {0}" , obj2.MyValue);

            // Bạn có thể thay đổi giá trị của trường.
            obj2.MyValue = "Value2-2";

            Console.ReadLine();
        }

    }
}
```

Ví dụ readonly & static readonly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
    class ConstFieldExample
    {
        // Một trường hằng số, giá trị của nó được xác định sẵn tại thời điểm
biên dịch.
        // Trường const không cho phép gán giá trị mới.
        // Chú ý: Trường const (Đã là static).
        public const int MY_VALUE = 100;

        // Một trường tĩnh và readonly.
        // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán 1 lần trong cấu
tử tĩnh.
        public static readonly DateTime INIT_DATE_TIME1 = DateTime.Now;

        // Một trường readonly.
        // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán một lần tại cấu
tử (không tĩnh).
        public readonly DateTime INIT_DATE_TIME2 ;

        public ConstFieldExample()
        {
            // Gán giá trị cho trường readonly (Chỉ được phép gán 1 lần).
            INIT_DATE_TIME2 = DateTime.Now;
        }
    }
}
```

Phương thức (Method)

Phương thức (Method)

- **Phương thức thông thường.**
- **Phương thức tĩnh**
- **Phương thức sealed. (Sẽ được đề cập trong phần thừa kế của class).**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class MethodSample
  {
      public string text = "Some text";

      // Cấu tử mặc định.
      // Nghĩa là cấu tử không có tham số.
      public MethodSample()
      {

      }

      // Đây là một phương thức trả về kiểu String.
      // Phương thức này không có tham số
      public string GetText()
      {
          return this.text;
      }

      // Đây là một phương thức có 1 tham số String.
      // Phương thức này trả về void (Hay gọi là ko trả về gì)
      public void SetText(string text)
      {
          // this.text tham chiếu tới trường text.
          // phân biệt với tham số text.
          this.text = text;
      }

      // Đây là một phương thức tĩnh.
      // Trả về kiểu int, có 3 tham số.
      public static int Sum(int a, int b, int c)
      {
          int d = a + b + c;
```

```
            return d;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
    class MethodSampleTest
    {
        static void Main(string[] args)
        {
            // Tạo đối tượng MethodSample
            MethodSample obj = new MethodSample();

            // Các phương thức không tĩnh cần phải được gọi thông qua đối
tượng.
            // Gọi phương thức GetText()
            String text = obj.GetText();

            Console.WriteLine("Text = " + text);

            // Các phương thức không tĩnh cần phải được gọi thông qua đối
tượng.
            // Gọi method SetText(String)
            obj.SetText("New Text");

            Console.WriteLine("Text = " + obj.GetText());

            // Các phương thức tĩnh cần phải được gọi thông qua Class.
            int sum = MethodSample.Sum(10, 20, 30);

            Console.WriteLine("Sum  10,20,30= " + sum);

            Console.ReadLine();
        }
    }
}
```

Note: - Các phương thức không tĩnh cần phải được gọi thông qua đối tượng.
- Các phương thức tĩnh cần phải được gọi thông qua Class.

Thừa kế trong C#

CSharp cho phép viết class mở rộng từ một class khác. Class mở rộng từ một class khác được gọi là class con. Class con có được thừa kế các trường, thuộc tính và các method từ class cha.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  // Mô phỏng một lớp động vật.
  class Animal
  {
      public Animal()
      {

      }

      public void Move()
      {
          Console.WriteLine("Move ...!");
      }

  }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class Cat : Animal
  {

      public   void Say()
      {
          Console.WriteLine("Meo");
      }

      // Một method của class Cat.
      public void Catch()
      {
          Console.WriteLine("Catch Mouse");
      }
  }
```

```
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  // Con kiến
  class Ant : Animal
  {
  }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
  class AnimalTest
  {

      static void Main(string[] args)
      {

          // Khai báo một đối tượng Cat.
          Cat tom = new Cat();

          // Kiểm tra xem 'tom' có phải là đối tượng Animal ko.
          // Kết quả rõ ràng là true.
          bool isAnimal = tom is Animal;

          // ==> true
          Console.WriteLine("tom is Animal? " + isAnimal);

          // Gọi method Catch
          tom.Catch();

          // ==> Meo
          // Gọi vào method Say() của Cat.
          tom.Say();

          Console.WriteLine("--------------------");

          // Khai báo một đối tượng Animal
```

```
            // Khởi tạo đối tượng thông qua cấu tử của Cat.
            Animal tom2 = new Cat();

            // Gọi method Move()
            tom2.Move();

            Console.WriteLine("--------------------");


            // Thông qua cấu tử của class con, Ant.
            Ant ant = new Ant();

            // Gọi method Move() thừa kế được từ Animal.
            ant.Move();

            Console.ReadLine();
        }
    }
}
```