Microsoft®
Visual C#®

CHUYÊN ĐỀ NGÔN NGỮ LẬP TRÌNH 1

Người biên soạn: Hồ Quang Thái (MSCB: 2299)
BM. Công Nghệ Phần Mềm, Khoa CNTT&TT
Email: hqthai@cit.ctu.edu.vn
Số tín chỉ: 2 (20 LT + 20TH)

Chương 2

NGÔN NGỮ LẬP TRÌNH C#

Ngôn ngữ lập trình C#

- *Câu lệnh nhập xuất cơ bản trên Console*
- *Từ khóa*
- *Chú thích*
- *Biến và biểu thức*
- *Kiểu dữ liệu*
- *Các toán tử*
- *Chuỗi*
- *Mảng*
- *Cấu trúc điều kiện*
- *Cấu trúc lặp*
- *Break và Continue*
- *Phương thức*

Từ khóa (Keyword)

- Là từ dành riêng của ngôn ngữ.
- Mỗi từ khóa có 1 ý nghĩa nhất định

Từ khóa (Keyword)

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

Các câu lệnh nhập xuất cơ bản

- Cho phép nhận giá trị từ bàn phím:
  - *Console.Read()*: nhận 1 ký tự từ bàn phím
  - *Console.ReadLine()*: nhận 1 chuỗi ký tự từ bàn phím.

```
class Program
{
    static void Main(string[] args)
    {
        string emp_Name;
        Console.Write("Nhap vao ho ten: ");
        emp_Name = Console.ReadLine();
        Console.WriteLine("Ho ten vua nhap {0}",emp_Name);
        Console.ReadLine();
    }
}
```

```
Nhap vao ho ten: Roger Federer
Ho ten vua nhap Roger Federer
```

Các câu lệnh nhập xuất cơ bản

- 2 phương thức dùng để xuất giá trị của biểu thức lên màn hình:

```
Console.Write()
Console.WriteLine()
```

- Tham số của 2 phương thức này là 1 biểu thức có kiểu bất kỳ

Thí dụ:

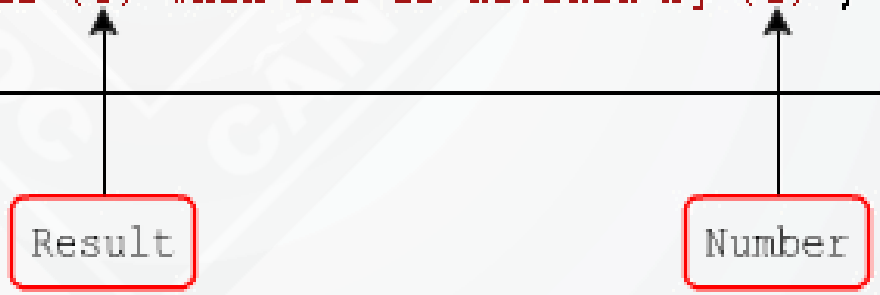
```
Console.Write("Hello World !!");
Console.WriteLine("Hello World !!");
```

Giữ chỗ (Placeholders)

- Tổng quát, các phương thức *Write()* và *WriteLine()* có thể có nhiều hơn 1 tham số:
  - Tham số đầu tiên là 1 chuỗi có những điểm đánh dấu trong cặp {} chỉ vị trí của giá trị các biểu thức là những tham số kế tiếp sẽ hiển thị.
  - Các tham số kế tiếp là các biểu thức.
  - Các điểm đánh dấu được viết bắt đầu là 0. Chẳng hạn: {0}, {1}.

Ví dụ về giữ chỗ

```
int number, result;
number = 5;
result = 100 * number;
Console.WriteLine("Result is {0} when 100 is multiplied by {1}",
result, number);
result = 150 / number;
Console.WriteLine("Result is {0} when 150 is divided by {1}",
  result, number);
```



Chú thích

- Cũng giống **C** và **C++**, **C#** dung ký tự **//** và **/* */** để ghi chú thích trong chương trình.
- Khi biên dịch thì các phần chú thích sẽ được bỏ qua.

```
// Chú thích một dòng
/* Chú thích
nhiều dòng */
```

Chú thích

- Thêm vào đó, có 2 chú thích dành cho phần mô tả tài liệu. Người ta dùng **///** cho chú thích tài liệu một dòng và **/** ***** ***/** cho chú thích tài liệu nhiều dòng.

```
/// <summary>Class level documentation.</summary>
class MyApp
{
        /** <summary>Program entry point.</summary>
            <param name="args">Command line arguments.</param>
        */
        static void Main(string[] args){
                System.Console.WriteLine("Hello World");
        }
}
```

Biến

- Biến là đại lượng lưu trữ dữ liệu trong quá trình tính toán.
- Một biến phải được khai báo (*declare*) trước khi được sử dụng.

  Thí dụ:
  ```
  int myInt;
  ```

- Chúng ta có thể khởi tạo giá trị (*assign*) bằng toán tử gán =.

  Thí dụ:
  ```
  myInt = 10;
  int myInt = 10;
  int myInt = 10, myInt2 = 20, myInt3;
  ```

Quy tắc đặt tên

- Một tên chỉ có thể chứa ký tự, ký số và dấu gạch dưới.
- Bắt đầu phải là 1 ký tự hoặc là dấu gạch dưới.
- Tên không được trùng với từ khóa.
- Phân biệt ký tự hoa thường, do đó **Count** và **count** là 2 tên khác nhau.

Tên hợp lệ và không hợp lệ

Hợp lệ	Không hợp lệ
Employee	12A2
student	class
Name	Tom&Jerry
_EmpName	Lion King

Biểu thức (Expression)

- Là sự kết hợp giữa các toán hạng và toán tử nhằm tính toán ra 1 giá trị nhất định.
  - *Toán hạng*: biến, hằng (có tên hoặc trực tiếp).
  - *Toán tử*: các phép toán.
- Biểu thức là 1 phần của câu lệnh hoặc 1 biểu thức khác

Thí dụ: `Pi*R*R`

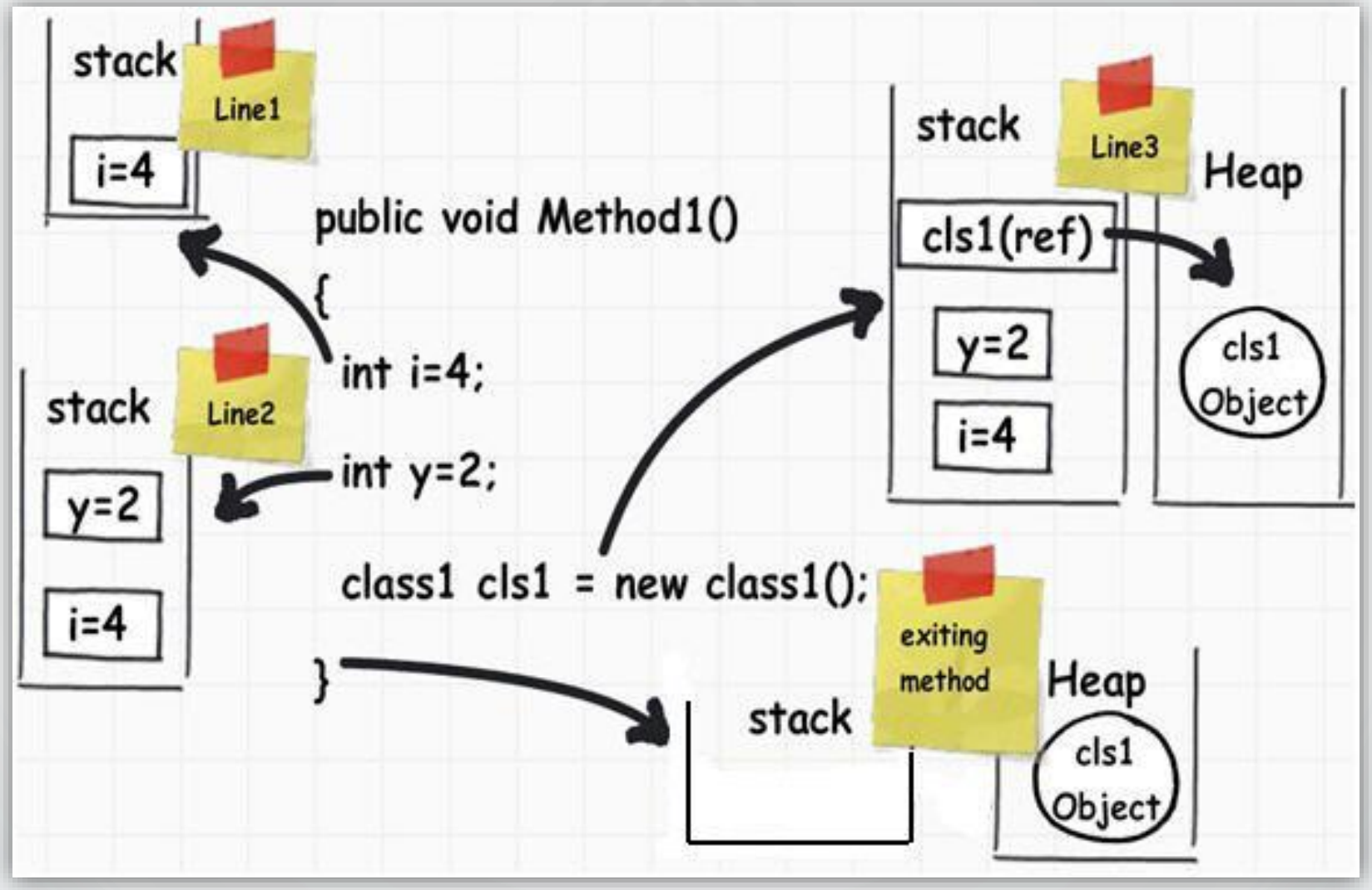
Kiểu dữ liệu

- Một kiểu dữ liệu là một tập hợp các giá trị và tập hợp các phép toán thao tác trên các giá trị đó.
- **C#** chia kiểu dữ liệu ra làm 2 loại:
  - ***Kiểu sơ cấp chuẩn*** *(Primitive Types)*
  - ***Kiểu tham chiếu*** *(Reference Types)*

Kiểu dữ liệu

- ***Kiểu giá trị***: biến có kiểu giá trị lưu giá trị thực sự trong ***stack***.
- ***Kiểu tham chiếu***: biến có kiểu tham chiếu lưu giá trị thực sự trong ***heap***, bên cạnh đó tham chiếu đến biến đó được ghi nhận trong ***stack***.
- Cả 2 loại kiểu giá trị hay kiểu tham chiếu có thể là kiểu có sẵn hoặc do người lập trình định nghĩa.

Kiểu dữ liệu


stack
Line1
i=4
public void Method1()
{
int i=4;
int y=2;
class1 cls1 = new class1();
}
stack
Line2
y=2
i=4
stack
Line3
Heap
cls1(ref)
y=2
i=4
cls1
Object
exiting
method
stack
Heap
cls1
Object

Các kiểu tham chiếu

- ***Object***: là kiểu cơ sở của mọi kiểu khác.
- ***String***: là kiểu tham chiếu có sẵn cho phép các biến kiểu này có thể lưu trữ dữ liệu chuỗi ký tự.
- ***Class***: là kiểu do người lập trình định nghĩa
- ***Delegate***: là kiểu do người lập trình định nghĩa cho phép các biến kiểu này tham chiếu đến một hay một số phương thức.
- ***Interface***: kiểu do người lập trình định nghĩa
- ***Array***: kiểu do người lập trình định nghĩa cho phép các biến kiểu này chứa các phần tử là những giá trị cùng kiểu

Các kiểu sơ cấp

Tên kiểu	bits	Ghi chú
sbyte	**8**	Số nguyên có dấu
byte	**8**	Số nguyên không dấu
ushort	**16**	
short	**16**	
uint	**32**	
int	**32**	
ulong	**64**	
long	**64**	
float	**32**	Số có dấu chấm động
double	**64**	
decimal	**128**	
char	**16**	Ký tự Unicode
bool	**4**	Giá trị logic (đúng,sai)

Kiểu số nguyên

- Có 4 kiểu số nguyên có dấu

```
sbyte myInt8 = 2; // -128 to +127
short myInt16 = 1; // -32768 to +32767
int myInt32 = 0; // -2^31 to +2^31-1
long myInt64 = -1; // -2^63 to +2^63-1
```

- Có 4 kiểu số nguyên không dấu

```
byte uInt8 = 0; // 0 to 255
ushort uInt16 = 1; // 0 to 65535
uint uInt32 = 2; // 0 to 2^32-1
ulong uInt64 = 3; // 0 to 2^64-1
```

- Số nguyên cũng có thể được gán giá trị hex

```
int myHex = 0xF; // hexadecimal (base 16)
```

Kiểu số thực

- Lưu trữ số thực có dấu chấm động

```
float myFloat = 3.14F; // chính xác 7 số thập phân
double myDouble = 3.14; // chính xác 15-16 số thập phân
decimal myDecimal = 3.14M; // chính xác 28-29 số thập phân
```

- Chuyển đổi kiểu

```
myFloat = (float)myDecimal;
```

- Số nguyên viết dưới dạng số mũ khoa học

```
myDouble = 3e2; // 3*10^2 = 300
```

CHAR, BOOL, PHẠM VI BIẾN

- Kiểu Char mang ký tự giá trị Unicode

```
char c = '3'; // Unicode char
```

- Kiểu Bool

```
bool b = true; // bool value
```

- Phạm vi biến, quy định biến trong và biến ngoài được quy định như **C** và **C++**

Chuyển đổi kiểu - Lớp Convert

```
class Program
{
    static void Main(string[] args)
    {
        string emp_Name;
        int emp_Age;
        double emp_Salary;

        Console.Write("Nhap vao ho ten: ");
        emp_Name = Console.ReadLine();
        Console.Write("Nhap vao tuoi: ");
        emp_Age = Convert.ToInt32(Console.ReadLine());
        Console.Write("Nhap vao luong: ");
        emp_Salary = Convert.ToDouble(Console.ReadLine());

        Console.WriteLine("Ho ten {0}; tuoi {1}; luong {2}",emp_Name,emp_Age, emp_Salary);
        Console.ReadLine();
    }
}
```

```
Nhap vao ho ten: Roger Federer
Nhap vao tuoi: 29
Nhap vao luong: 40000
Ho ten Roger Federer; tuoi 29; luong 40000
```

Toán tử

- Toán tử số học:

```
float x = 3 + 2; // 5 // Cộng
x = 3 - 2; // 1 // Trừ
x = 3 * 2; // 6 // Nhân
x = 3 / 2; // 1 // Chia
x = 3 % 2; // 1 // Phép chưa lấy dư
```

- Lưu ý với phép chia với các số nguyên sẽ cho kết quả là một số nguyên. Để có thể chia đúng ta nên chuyển đổi kiểu

```
x = 3 / (float)2; // 1.5
```

Toán tử kết hợp

- Các toán tử số học có thể được viết gọn lại thành dạng như sau :

```
int x = 0;
    x += 5; // x = x+5;
    x -= 5; // x = x-5;
    x *= 5; // x = x*5;
    x /= 5; // x = x/5;
    x %= 5; // x = x%5;
```

Toán tử tăng giảm

- Toán tử dùng để tăng hoặc giảm 1 đơn vị

```
x++; // x = x+1;
x--; // x = x-1;
```

- Cả 2 toán tử này đều có thể sử dụng trước và sau một biến.

```
x++; // tăng sau
x--; // giảm sau
++x; // tăng trước
--x; // giảm trước
```

- ***Ví dụ:***

```
x = 5; y = x++; // y=5, x=6
x = 5; y = ++x; // y=6, x=6
```

Toán tử so sánh

- Toán tử luôn trả về giá trị đúng hoặc sai của một biểu thức

```
bool x = (2 == 3); // false // equal to
x = (2 != 3); // true // not equal to
x = (2 > 3); // false // greater than
x = (2 < 3); // true // less than
x = (2 >= 3); // false // greater than or equal to
x = (2 <= 3); // true // less than or equal to
```

- Toán tử logic, là toán tử kết hợp **&&** và **||**

```
bool x = (true && false); // false // logical and
x = (true || false); // true // logical or
x = !(true); // false // logical not
```

Toán tử bitwise

- Thực hiện tính toán trên *từng bit nhị phân*

```
int x = 5 & 4; // 101 & 100 = 100 (4) // and
x = 5 | 4; // 101 | 100 = 101 (5) // or
x = 5 ^ 4; // 101 ^ 100 = 001 (1) // xor
x = 4 << 1; // 100 << 1 =1000 (8) // left shift
x = 4 >> 1; // 100 >> 1 = 10 (2) // right shift
x = ~4; // ~00000100 = 11111011 (-5) // invert
```

- Cách viết rút gọn

```
int x=5; x &= 4; // 101 & 100 = 100 (4) // and
x=5; x |= 4; // 101 | 100 = 101 (5) // or
x=5; x ^= 4; // 101 ^ 100 = 001 (1) // xor
x=5; x <<= 1; // 101 << 1 =1010 (10)// left shift
x=5; x >>= 1; // 101 >> 1 = 10 (2) // right shift
```

Thứ tự ưu tiên của các toán tử

Thứ tự	Toán tử	Thứ tự	Toán tử
1	++ -- ! ~	7	&
2	* / %	8	^
3	+ -	9	|
4	<< >>	10	&&
5	< <= > >=	11	||
6	== !=	12	=op=

Ví dụ:

```
bool x = 2+3 > 1*4 && 5/5 == 1
```

Chuỗi

- Khai báo và khởi tạo biến string

```
string a = "Hello";
```

- Để nối chuỗi, chúng ta dung toán tử + hoặc +=

```
string b = a + " World"; // Hello World
a += " World" ; // Hello World
string c
      = "Hello " +
                  "World";
```

- Dùng '\n' để xuống dòng

```
string c = "Hello\nWorld";
```

Chuỗi

Ký tự	Ý nghĩa	Ký tự	Ý nghĩa
\n	Xuống dòng	\f	Sang trang
\t	Cách dòng tab	\a	Âm thanh thông báo
\v	Cách hàng tab	\’	Dấu nháy đơn
\b	~phím backspace	\”	Dấu nháy đôi
\r	Trở về đầu dòng	\\	Dấu \
\0	Ký tự rỗng	\uFFFFF	Ký tự unicode

- Ta có thể sử dụng ký @ trong chuỗi chứa đường dẫn

```
string e = "c:\\Windows\\System32\\cmd.exe";
string f = @"c:\Windows\System32\cmd.exe";
```

Chuỗi

- So sánh chuổi

```
bool c = (a == b); // true
```

- Các hàm con thường sử dung trong lớp **String**

```
string a = "String";
string b = a.Replace("i","o"); // Strong
b = a.Insert(0, "My "); // My String
b = a.Remove(0, 3); // ing
b = a.Substring(0, 3); // Str
b = a.ToUpper(); // STRING
int i = a.Length; // 6
```

- Lớp **StringBuilder**

```
System.Text.StringBuilder sb = new
System.Text.StringBuilder("Hello");
sb.Append(" World"); // Hello World
sb.Remove(0, 5); // World
sb.Insert(0, "Bye"); // Bye World
string s = sb.ToString(); // Bye World
```

MẢNG

- Mảng là 1 tập hợp các phần tử có cùng kiểu dữ liệu, mỗi phần tử có một vị trí xác định.

Khai báo và khởi tạo mảng

- Do mảng thuộc kiểu tham chiếu nên ta cần khai báo và khởi tạo đối tượng mảng.
- Khai báo:

```
int[] x; // not int x[]
```

- Khởi tạo đối tượng mảng:

```
int[] x = new int[3];
```

- Truy xuất giá trị:

```
x[0] = 1;
x[1] = 2;
x[2] = 3;
```

- Ta cũng có thể vừa khai báo vừa gán giá trị

```
int[] y = new int[] { 1, 2, 3 };
int[] z = { 1, 2, 3 };
```

Mảng hai chiều

- Mảng 2 chiều là mảng có cùng độ dài ở các phần tử

```
string[,] x = new string[2, 2];
```

- Giống như mảng một chiều, các phần tử có thể được khởi tạo trước hoặc sau khi khai báo

```
x[0, 0] = "00" ; x[0, 1] = "01" ;
x[1, 0] = "10" ; x[1, 1] = "11" ;

string[,] y = { { "00", "01" }, { "10", "11" } };
```

Mảng nhiều chiều

- Mảng nhiều chiều là một mảng mà mỗi phần tử là một mảng khác

```
string[][] a = new string[2][];
a[0] = new string[1]; a[0][0] = “00”;
a[1] = new string[2]; a[1][0] = “10”;
a[1][1] = “11”;
```

- Bạn cũng có thể vừa khai báo và khởi tạo

```
string[][] b = { new string[] { "00" },
                 new string[] { "10", "11" } };
```

- Ngoài ra, bạn cũng có thể thêm nhiều dấu phẩy hơn trong mảng 2 chiều để có nhiều chiều hơn nếu cần

ArrayList

- **ArrayList** được tạo ra nhằm giải quyết các hạn chế của Array. **ArrayList** hỗ trợ sắp xếp và các hàm tiện ích khác.
- Dữ liệu thành viên trong **ArrayList** phải là kiểu tham chiếu

```
// Khai báo và khởi tạo một ArrayList
ArrayList a = new ArrayList();

a.Add("Hi"); // Thêm một phần tử
a.Insert(0, "Hello"); // Thay đổi phần từ đầu tiên
a.RemoveAt(0); // Xóa phần tử đầu tiên
a.Add("Hello World"); // Thêm lại một phần tử
String s = a[0]; // Hello World
```

Câu lệnh if

- Câu lệnh **if** sẽ chỉ thực hiện các câu lệnh bên trong nếu điều kiện là đúng.
- Biểu thức điều kiện có thể bao gồm các *toán tử so sánh* và *toán tử logic*.
- Sử dụng **else if** khi muốn tiếp tục xét một trường hợp với biểu thức điều kiện khác.
- Sử dụng **else** để xét những trường còn lại.

Ví dụ về câu lệnh IF

```
int x = new System.Random().Next(3);
// Tạo một số ngẫu nhiên từ 0 đến 2
if (x < 1) {
        System.Console.Write(x + " < 1"); }
else if (x > 1) {
        System.Console.Write(x + " > 1"); }
else {
        System.Console.Write(x + " == 1");
}
```

Câu lệnh switch

- Câu lệnh **switch** kiểm tra trường hợp điều kiện dựa trên một biến *kiểu số nguyên* hoặc *chuỗi* trong một nhóm các nhãn **case**

```
int x = new System.Random().Next(3); // gives 0, 1 or 2
switch (x) {
        case 0: System.Console.Write(x + " is 0"); break;
        case 1: System.Console.Write(x + " is 1"); break;
        default:System.Console.Write(x + " is 2"); break;
}
```

Câu lệnh goto

- Trong trường hợp muốn nhảy đến một trường hợp khác. Chúng ta dùng lệnh **goto** để nhảy.

```
case 0: goto case 1;
```

- Bạn có thể nhảy để một nhãn đã được đặt trước.

```
goto myLabel;
// ...
myLabel:
```

- tuy nhiên việc sử dụng **goto** để nhảy đến một nhãn không được khuyến khích vì nó có thể gây khó hiểu trong dòng thực thi.

Toán tử ?:

- Toán tử này nhằm thay thế câu lệnh **if else** thành một dạng rút gọn hơn.

```
// Giá trị giữa 0.0 và 1.0
double x = new System.Random().NextDouble();

x = (x < 0.5) ? 0 : 1; // ternary operator (?:)
```

Vòng lặp while

- Vòng lặp **while** thực thi các dòng lệnh trong **{...}** khi ***điều kiện đúng***.

```
int i = 0;
while (i < 10) {
        System.Console.Write(i++);
}
// 0-9
```

Vòng lặp do-while

- Vòng lặp **do-while** thực thi các dòng lệnh trong **do {...}** một lần, sau đó sẽ kiểm tra điều kiện trong **while** xem có tiếp tục thực hiện lại công việc hay không.

```
int j = 0;
do {
        System.Console.Write(j++);
}
while (j < 10);
// 0-9
```

Vòng lặp for

- Vòng lặp **for** cho phép lặp số vòng lặp xác định. Cho phép khai báo biến đếm dành cho mỗi lần lặp.

```
for (int k = 0; k < 10; k++) {
        System.Console.Write(k);
}
// 0-9
```

Vòng lặp for each

- Cho phép đi qua 1 tập hợp các phần tử

```
int[] a = { 1, 2, 3 };
foreach (int n in a) {
        System.Console.Write(n); // 123
}
```

- Chú ý là các phần tử phải là **read-only** và dòng lặp **foreach** không thể thay đổi các thành phần trong một mảng

Câu lệnh break

- Cho phép thoát khỏi vòng lặp không cần kiểm tra điều kiện của vòng lặp

```
int n=11;
for (int i=2;i<=n-1;i++)
        if (n%i!=0) break;

if (i==n) Console.WriteLine("{0} là số nguyên tố", n);
else Console.WriteLine("{0} không là số nguyên tố", n);

Console.ReadLine();
```

Câu lệnh continue

- Câu lệnh continue cho phép kết thúc lần lặp hiện hành và chuẩn bị bắt đầu lần lặp kế tiếp.
- **Thí dụ**: Liệt kê các số chẵn từ 1 đến 10:

```
Console.WriteLine("Các số chẵn: ");
for (int i=1;i<=10;i++){
        if (i%2!=0) continue;
        Console.WriteLine(i + " ");
}

Console.ReadLine();
```

Kiểu nullable và toán tử ??

- Là kiểu có thể giữ giá trị **null**.
- **Thí dụ**:

```
bool? b = null;
```

- Toán tử **??** trả lại giá trị bên trái nếu nó không **null**, nếu nó **null** thì trả lại giá trị bên phải.

```
int? i = null;
int j = i ?? 0; // 0
```

- Một biến kiểu **nullable** không nên chuyển sang kiểu **non-nullable** vì có thể xảy ra lỗi trong quá trình thực thi.

```
int j = (int)i; // run-time error
```

Giá trị mặc định

- Giá trị mặc định của kiểu tham chiếu là **null**.
- Giá trị mặc định của kiểu sơ cấp:
  - *Kiểu số*: **0**
  - *Kiểu ký tự*: **\0000**
  - *Kiểu bool*: **false**;
- Giá trị mặc định sẽ *tự động khởi tạo khi biên dịch*. Tuy nhiên, nên khởi tạo giá trị cho biến một cách rõ ràng để code trở nên dễ hiểu hơn.
- Biến cục bộ sẽ không được gán giá trị mặc định. Thay vào đó, trình biên dịch sẽ yêu cầu lập trình viên phải khai báo giá trị cho biến cục bộ.

Giá trị mặc định

```
class MyApp
{
        int x; // trường được khởi tạo giá trị mặc định 0
        int dummy()
        {
                int x;
                // biến cục bộ phải được khởi tạo
                   trước khi sử dụng
        }
}
```

Phương thức

- Tất cả các phương thức phải nằm trong một lớp, **C#** không có phương thức toàn cục bên ngoài.
- Từ khóa **void** được sử dụng cho những phương thức không trả về giá trị nào.

```
class MyApp {
        void MyPrint(){
                System.Console.Write("Hello World");
        }
}
```

Gọi phương thức

- Để gọi một phương thức, cần có một *thể hiện* (*instance*) của lớp ***MyApp*** bằng từ khóa **new**
- Dùng dấu chấm (.) phía sau thể hiện để gọi phương thức của lớp đối tượng đó.

```
class MyApp {
        static void Main(){
                MyApp m = new MyApp();
                m.MyPrint(); // Hello World
        }
        void MyPrint(){
                System.Console.Write("Hello World");
        }
}
```

Tham số

- Tham số truyền vào phương thức có kiểu dữ liệu xác định, các tham số cách nhau bởi dấu phẩy (,)

```
void MyPrint(string s1, string s2)
{
        System.Console.Write(s1 + s2);
}
```

```
static void Main()
{
        MyApp m = new MyApp();
        m.MyPrint("Hello", " World"); // Hello World
}
```

Từ khóa params

- Để truyền một số các biến có kiểu xác định, ta có thể dùng từ khóa **params** vào tham số cuối cùng trong danh sách.

```
void MyPrint(params string[] s)
{
        foreach (string x in s)
                System.Console.Write(x);
}
```

Nạp chồng phương thức

- Chúng ta có thể tạo ra nhiều phương thức có *cùng tên* nhưng *khác biến và kiểu trả về*, chúng được gọi là nạp chồng phương thức.
- ***Ví dụ***: Hàm System.Console.Write() có 18 hàm.

```
void MyPrint(string s)
{
        System.Console.Write(s);
}
void MyPrint(int i)
{
        System.Console.Write(i);
}
```

Tham số tùy chọn (Optional parameters)

- **C# 4.0** về sau, tham số có thể được khai báo theo kiểu tùy chọn với giá trị mặc định trong phương thức.
- Khi được gọi, những tham số tùy chọn có thể bỏ qua để sử dụng giá trị mặc định.

```
class MyApp {
        void MySum(int i, int j = 0, int k = 0){
                System.Console.Write(1*i + 2*j + 3*k);
        }

        static void Main(){
                new MyApp().MySum(1, 2); // 5
        }
}
```

Đối số gọi tên (Named arguments)

- **C# 4.0** về sau cho phép một đối số có thể truyền sử dụng tên tương ứng với tham số đó.

```
static void Main() {
        new MyApp().MySum(1, k: 2); // 7
}
```

- Ta cũng có thể xác định tham số tùy chọn mà không cần xác định giá trị trước.

Lệnh return

- Một phương thức có thể trả lại một giá trị.
- Từ khóa **return** sẽ trả lại giá trị tương ứng với kiểu đã xác định trong phương thức.

```
string GetPrint()
{
    return "Hello";
}
```

```
static void Main(){
  MyApp m = new MyApp();
  System.Console.Write(m.GetPrint());
  // Hello World
}
```

- Lệnh **return** cũng có thể được sử dụng để thoát khi hết khối lệnh.

```
void MyMethod(){
    return;
}
```

Truyền giá trị

- Thông thường, phương thức chỉ có thể truyền giá trị. Khi đó, giá trị thực được truyền vào phương thức thay cho biến.
- Biến sẽ không được thay đổi sau khi truyền bằng giá trị.

```
void Set(int i) { i = 10; }

static void Main(){
        MyApp m = new MyApp();
        int x = 0; // value type
        m.Set(x); // pass value of x
        System.Console.Write(x); // 0
}
```

Truyền tham chiếu

- Truyền khi dữ liệu truyền là kiểu tham chiếu.

```
void Set(int[] i) {
        i = new int[] { 10 };
}

static void Main()
{
        MyApp m = new MyApp();
        int[] y = { 0 }; // reference type
        m.Set(y); // pass object reference
        System.Console.Write(y[0]); // 10
}
```

Từ khóa ref

- Một kiểu giá trị có thể được truyền bằng từ khóa **ref**.
- Cả phương thức lẫn lời gọi đều phải khai báo.

```
void Set(ref int i) {
        i = 10;
}

static void Main(){
        MyApp m = new MyApp();
        int x = 0; // value type
        m.Set(ref x); // pass reference to value type
        System.Console.Write(x); // 10
}
```

Từ khóa out

- Đôi khi bạn muốn truyền một biến chưa được khởi tạo và sẽ khởi tạo trong phương thức. Trong trường hợp này từ khóa **out** sẽ được sử dụng. Nó giống như **ref** ngoại trừ việc trình biên dịch cho phép nó không cần khởi tạo trước.

```
void Set(out int i) { i = 10; }
static void Main()
{
        MyApp m = new MyApp();
        int x; // value type
        m.Set(out x); // pass reference to unset value type
        System.Console.Write(x); // 10
}
```