

CHƯƠNG 2: LẬP TRÌNH C# CƠ BẢN

Thời gian: 180 phút

Người trình bày: HỒ LÊ VIỆT NIN

Email: holevietnin@duytan.edu.vn

Điện thoại: 0905455246



NỘI DUNG

1. Những nội dung cơ bản
2. Các lệnh trong C#
3. Mảng và tập hợp trong C#



NỘI DUNG CƠ BẢN

- ❖ Cấu trúc chương trình C#
- ❖ Kiểu dữ liệu, từ khoá, định danh biến, hằng...
- ❖ Chuyển đổi kiểu
- ❖ Console I/O
- ❖ Tham số ref, out, param

Video:

<https://www.youtube.com/watch?v=2VkYpsKCzGg>



Cấu trúc chương trình C#

```
using <namespace sử dụng>

...
namespace <Tên Namespace>
{
    [Khóa truy xuất] class <Tên lớp>
    {
        public static void Main()
        {
            ...
        }

        // thành viên khác ...
    }

    // lớp khác ...
}
```

.cs

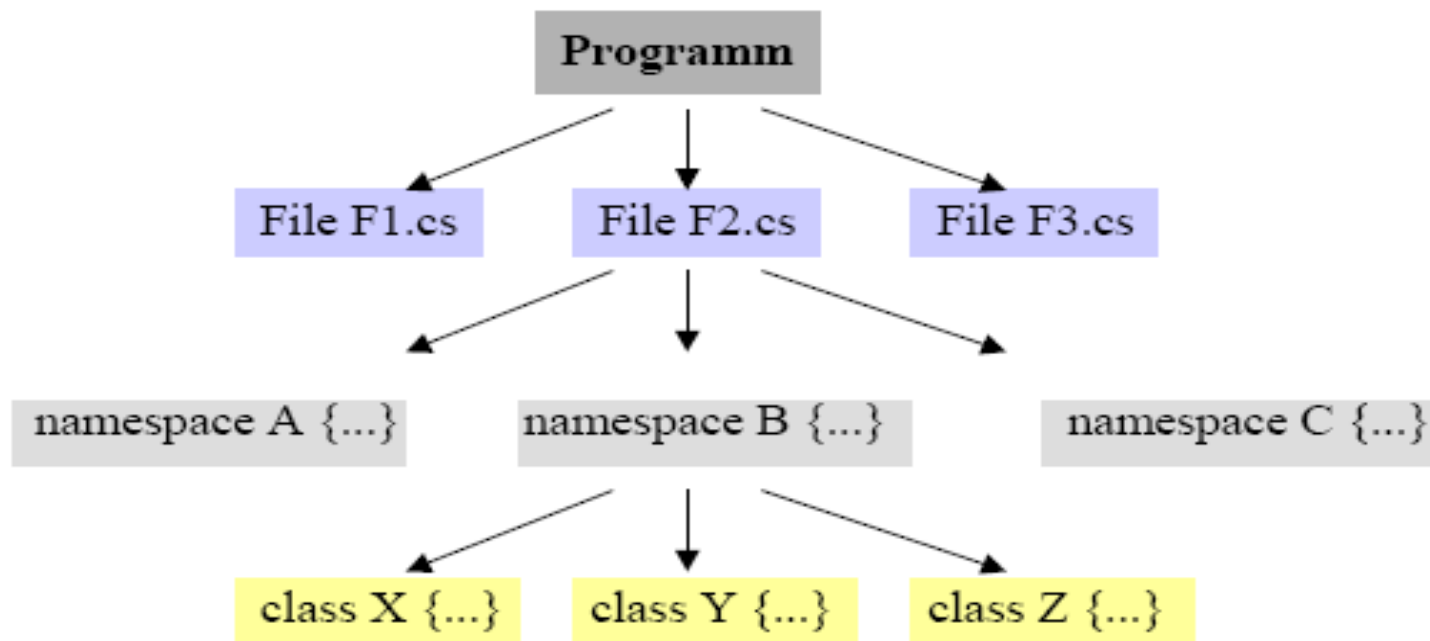


Cấu trúc chương trình C#

- **using**: làm code gọn hơn, ko cần phải dùng tên của namespace
 - using System.
- namespace của chương trình: ko bắt buộc
- **class**: tối thiểu có 1 lớp chứa hàm entry point Main của chương trình
- public **static** void **Main**(): hàm entry point của chương trình C#



Cấu trúc chương trình C#



- Nếu ko có namespace \Rightarrow namespace mặc định ko tên
- Namespace có thể chứa: struct, interface, delegate, enum
- Trường hợp đơn giản nhất: 1 lớp, 1 file cs và namespace mặc định



Kiểu dữ liệu

- Bao gồm

- Lớp đối tượng
- Ký tự, chuỗi
- Số nguyên có dấu
- Số nguyên không dấu
- Số thực
- Kiểu logic

object

char, string

sbyte, short, int, long

byte, ushort, uint, ulong

float, double, decimal

bool



Kiểu dữ liệu

- Là alias của lớp dữ liệu trong .NET
 - **s**tring System.**S**tring
 - **i**nt System.**I**nt32
 - **o**bject System.**O**bject



Kiểu dữ liệu

- Sử dụng kiểu dữ liệu
 - Định nghĩa trước (C#)
 - Built-in value type: int, long, string, object...
 - Chương trình định nghĩa (tham chiếu)
 - Class, struct, enum...
 - Person, Student, Employee...

Data Type

Built-in

User defined



Kiểu dữ liệu giá trị

Name	CTS Type	Size	Range
sbyte	System.SByte	8	-128..127
short	System.Int16	16	(-32768 .. 32767)
int	Sytem. Int32	32	$-2^{31}..2^{31}-1$
long	Sytem. Int64	64	$-2^{63}..2^{63}-1$
byte	System.SByte	8	0..255
ushort	System.UInt16	16	(0 .. 65535)
uint	System.UInt32	32	$0..2^{32}-1$
ulong	System.UInt64	64	$0..2^{64}-1$
float	System.Single	32	xấp xỉ từ 3,4E - 38 đến 3,4E+38
double	System.Double	64	1,7E-308 đến 1,7E+308
decimal	System.Decimal	128	Có độ chính xác đến 28 con số
bool	System.Boolean		Kiểu true/false
char	System.Char	16	Ký tự unicode



Kiểu dữ liệu tham chiếu

- object: System.Object
 - Kiểu dữ liệu gốc, cha của tất cả các kiểu dữ liệu trong C#

```
object o = new object();
```

- string: System.String
 - Chuỗi ký tự Unicode

```
string s1 = "Hutech";  
string s2 = "Hi";  
string s = s1 + s2;
```



Kiểu giữ liệu tham chiếu

```
using System;
```

```
class StringExample
```

```
{
```

```
    public static int Main()
```

```
    {
```

```
        string s1 = "a string";
```

```
        string s2 = s1;
```

```
        Console.WriteLine("s1 is " + s1);
```

```
        Console.WriteLine("s2 is " + s2);
```

```
        s1 = "another string";
```

```
        Console.WriteLine("s1 is now " + s1);
```

```
        Console.WriteLine("s2 is now " + s2);
```

```
        return 0;
```

```
    }
```

```
}
```

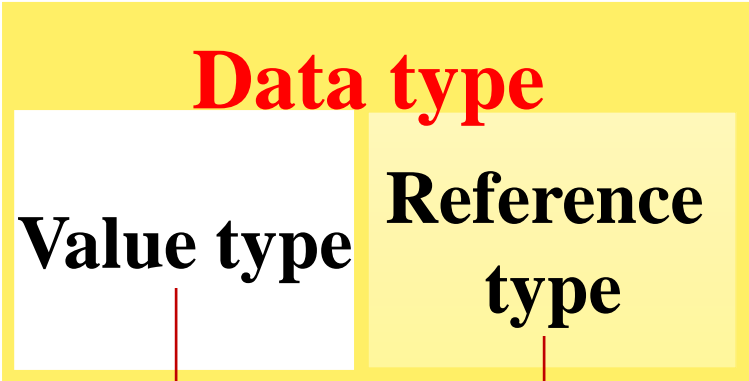


```
s1 is a string  
s2 is a string  
s1 is now another string  
s2 is now a string  
-
```



Phân loại kiểu dữ liệu

**Phân loại theo cách thức
lưu trữ dữ liệu**



**int num;
long count;**

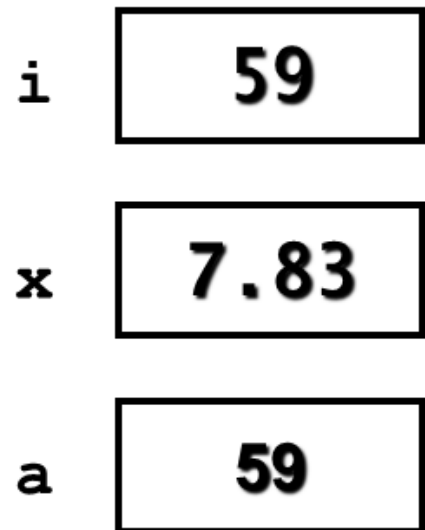
**Object obj = new Object();
String str = “reference type”;**



Value Type

- Chứa giá trị trực tiếp
- Không thể null
 - Phải chứa giá trị xác định
- Bao gồm
 - Primitive type
 - double, char, int, float
 - Enum
 - struct

```
int i = 59;  
double x = 7.83;  
int a = i;
```

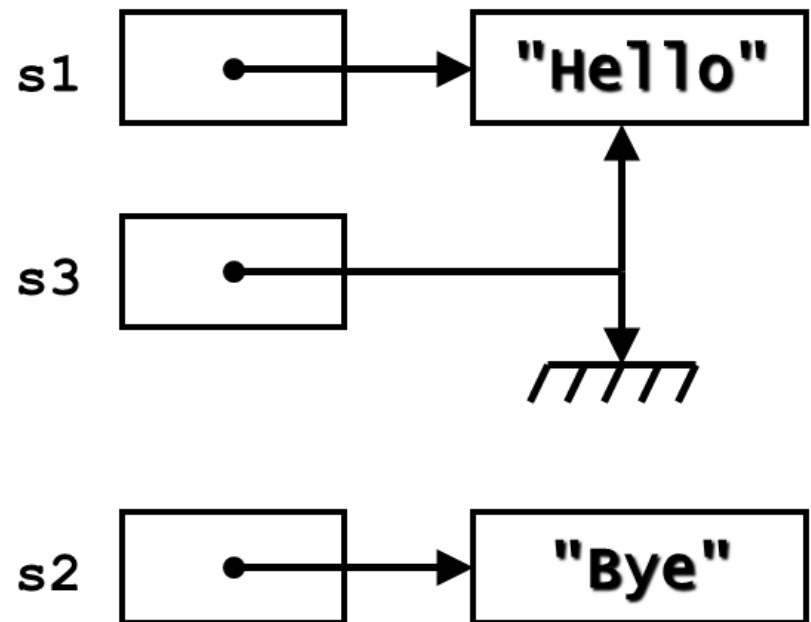




Reference type

- Chỉ tới nơi chứa dữ liệu
- Có thể **null**
 - **null**: không chỉ tới bất kỳ đâu
- Bao gồm
 - Lớp (class)
 - **string, object**
 - Giao diện (interface)
 - Mảng (array)
 - Đại diện (delegate)

```
string s1 = "Hello"  
string s2 = "Bye"  
string s3;  
s3 = s1;
```





Value type vs. Reference type

Characteristic	Value type	Reference type
Variable hold	Value	Reference
Allocated	Stack	Heap
Default	Zero	Null
Parameter	Copy value	Copy reference



Định danh (Identifier)

- Định danh: những từ được đặt ra để đại diện cho mọi thứ dùng trong chương trình
 - Khi đặt định danh: nên có tính gợi nhớ
- Tạo ra định danh mới
 - **HelloWorld, Program, Perform,...**
→ phải khai báo trước khi sử dụng
- Dùng định danh có sẵn
 - **Console, WriteLine, ReadLine,...**
→ phải chỉ ra nơi chứa định danh (namespace)



Định danh (Identifier)

- Bao gồm chữ cái, chữ số, ký tự gạch dưới
- Không được bắt đầu bằng chữ số
 - `Chuong_Trinh`, `x25`, `z`, `_abc`, `Xử Lý` → **hợp lệ**
 - `2abc`, `Chuong-Trinh`, `Xu Ly`, `class` → **không hợp lệ**
- Case-sensitive
 - `ChuongTrinh` và `chuongtrinh` là khác nhau
- Các định danh được khai báo trong cùng phạm vi (scope) không được trùng nhau
- Phải khác với từ khóa (*có thể dùng “@” trước từ khóa làm identifier*)



Từ khóa (Keyword)

Các từ khóa trong C# 2005

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



Hằng (Constant)

- Một hằng là một biến nhưng trị không thay đổi

`const int a = 100; // giá trị ko thể thay đổi`

- Hằng bắt buộc phải được gán giá trị lúc khai báo
- Trị của hằng có thể được tính toán vào lúc biên dịch
- Hằng bao giờ cũng static



Hằng (Constant)

- Ưu điểm
 - Chương trình dễ đọc, khắc phục những con số “*magic number*” trong code.
 - Chương trình dễ sửa hơn.
 - Tránh lỗi dễ dàng hơn, trình biên dịch sẽ báo lỗi nếu gán lại giá trị cho hằng



Hằng (Constant)

- Minh họa sử dụng hằng

```
public void Test1()
```

```
{
```

```
    int[] a = new int[100];
```

```
    Random rand = new Random();
```

```
    for(int i=0; i < 100; i++)
```

```
        a[i] = rand.Next(100);
```

```
    int k = 100;
```

```
    int j=0;
```

```
    while (j < 100 && a[j] != k)
```

```
        j++;
```

```
}
```



Định nghĩa hằng

```
public void Test2()
```

```
{
```

```
    const int MAXLEN = 100;
```

```
    const int MAXNUM = 100;
```

```
    int[] a = new int[MAXLEN];
```

```
    Random rand = new Random();
```

```
    for (int i = 0; i < MAXLEN; i++)
```

```
        a[i] = rand.Next(MAXNUM);
```

```
    int k = 100;
```

```
    int j = 0;
```

```
    while (j < MAXLEN && a[j] != k)
```

```
        j++;
```

```
}
```





readonly

const: phải được gán giá trị khi khai báo

readonly: ko cần khởi tạo trước, khi gán giá trị thì sau đó ko thay đổi được

```
class MyClass
{
    const int x = 5;
    readonly int y = 25;
    const int xx;
    readonly int yy;
    public MyClass()
    {
        yy = 24;
    }
    public void MyMethod()
    {
        x = 7;
        yy = 30;
    }
}
```



Chưa được khởi gán

Ko được thay đổi



Biến (Variable)

- Biến là nơi lưu dữ liệu của chương trình
- Dữ liệu của biến
 - Nằm trong bộ nhớ vật lý (physical RAM)
 - Có thể thay đổi giá trị
- Phải khai báo trước khi dùng
 - Identifier: tên để đại diện cho biến
 - Data type: dạng lưu trữ dữ liệu của biến

Data type **identifier**



Biến (Variable)

- Phạm vi (scope)
 - Được xác định bởi cặp dấu { và }
 - Có thể chứa phạm vi nhỏ hơn
- Vị trí khai báo biến
 - Trong thân phương thức: biến cục bộ
 - Trong thân lớp: thuộc tính
- Biến trong C# chỉ có tác dụng trong phạm vi mà nó được khai báo



Ép kiểu (Type cast)

- Chuyển đổi kiểu dữ liệu (ép kiểu): chuyển giá trị từ kiểu này sang kiểu khác (có liên quan với nhau)
- Ví dụ
 - Chuyển từ **int** qua **float** và ngược lại
- Có hai loại
 - Ép kiểu ngầm định
 - Ép kiểu tường minh



Ép kiểu ngầm định

- Do C# tự thực hiện
- Không bị mất thông tin
- Không cần lập trình viên can thiệp
- Xảy ra khi nào
 - Ép từ kiểu nhỏ sang kiểu lớn

```
int i = 59;  
float x = i;
```

- Sau khi gán thì hoàn toàn không bị mất dữ liệu vì bất cứ giá trị nào của int cũng thuộc về float.



Ép kiểu ngầm định

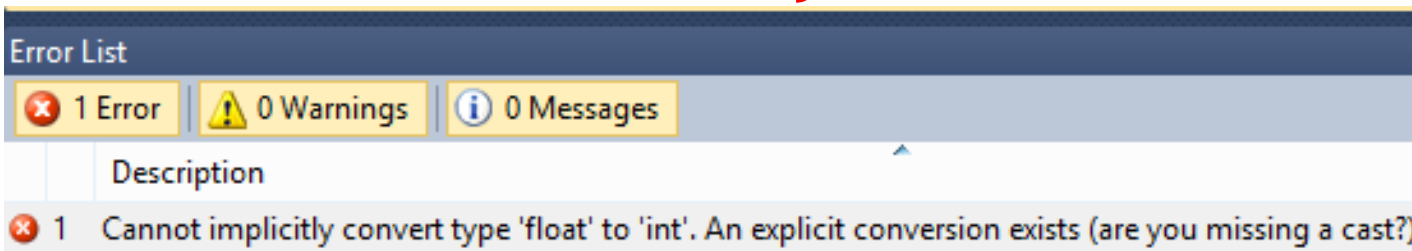
From	To
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long, ulong	float, double, decimal
float	double



Ép kiểu tường minh

- Ép từ kiểu lớn sang kiểu nhỏ

```
float i = 59;  
int x = i;
```



- Buộc phải dùng lệnh gán tường minh

```
float i = 59;  
int x = (int)i;
```



Lớp Convert

- Thường dùng khi cần chuyển đổi giữa các kiểu dữ liệu không liên quan với nhau

Convert.KiểuDữLiệu(*GiáTrịNguồn*)

❖ Ví dụ chuyển đổi từ kiểu chuỗi sang số thực

```
string s1 = "56.8";  
string s2 = "95";  
double x = Convert.ToDouble(s1); // x = 56.8  
int i = Convert.ToInt32(s2);      // i = 95  
byte j = Convert.ToByte(x); // j = 56, ít dùng
```



Phương thức Parse

- Thường dùng khi cần chuyển đổi giữa các kiểu dữ liệu không liên quan với nhau

KiểuDữLiệu.Parse(*GiáTrịNguồn*)

❖ Ví dụ chuyển đổi từ kiểu chuỗi sang số thực

```
string s1 = "56.8";  
string s2 = "95";  
double x = double.Parse(s1);      // x = 56.8  
int i = int.Parse(s2);             // i = 95
```



Câu lệnh nhập xuất

- Để đọc văn bản từ cửa sổ console
 - `Console.ReadLine()` giá trị trả về là string

```
string s = Console.ReadLine();  
Console.WriteLine(s);
```

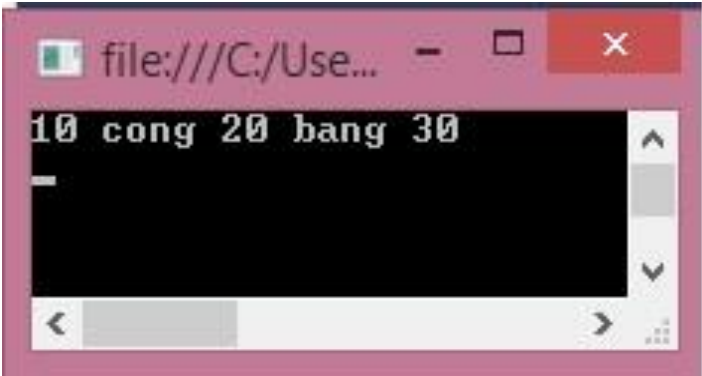
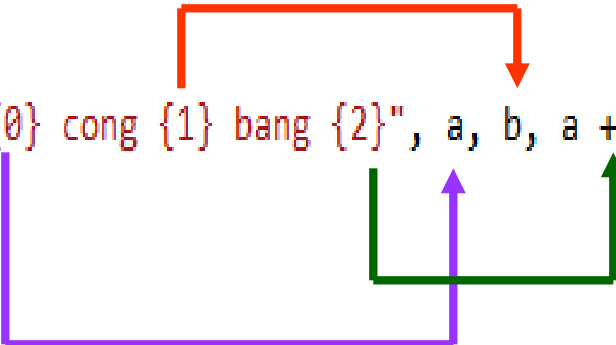
- Để xuất chuỗi ký tự thì ta dùng
 - `Console.Write()` / `Console.WriteLine()`



Câu lệnh nhập xuất

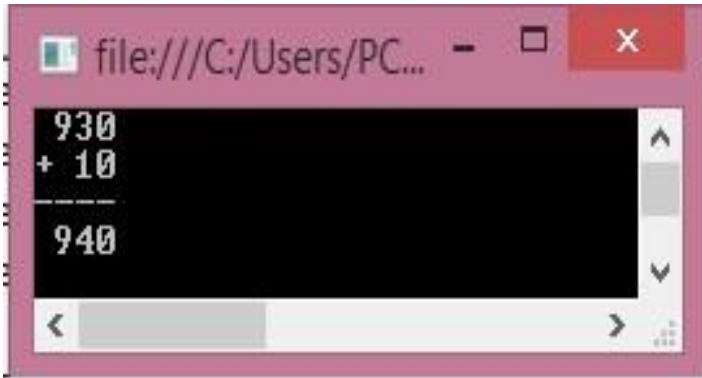
- Để xuất chuỗi dùng Console.WriteLine()

```
int a = 10;  
int b = 20;  
Console.WriteLine("{0} cong {1} bang {2}", a, b, a + b);
```



```
int a = 930;  
int b = 10;  
Console.WriteLine("{0,4}\n+{1,3}\n----\n{2,4}", a, b, a + b);
```

\n: ký tự xuống dòng





Boxing & Unboxing

- Kiểu giá trị có thể được chuyển thành kiểu đối tượng

- Boxing

```
int num1 = 100;  
char ch = 'c';
```

```
object o1 = num1, o2 = ch;
```

Boxing

- Unboxing

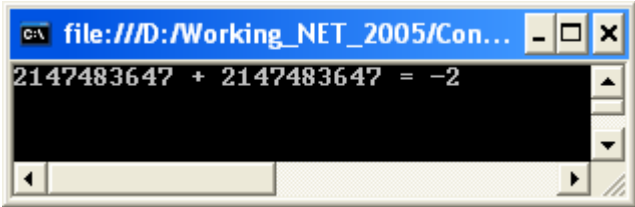
```
object o1 = 99, o2 = 'c';  
int number1 = (int)o1;  
char char1 = (char)o2;
```

Unboxing



checked & unchecked

```
int x = int.MaxValue;  
int y = int.MaxValue;  
int z = unchecked(x + y);
```



```
Console.WriteLine("{0} + {1} = {2}",  
    x.ToString(), y.ToString(), z.ToString());
```

```
int x = int.MaxValue;  
int y = int.MaxValue;  
int z = checked(x + y);
```



throws OverflowException

```
Console.WriteLine("{0} + {1} = {2}",  
    x.ToString(), y.ToString(), z.ToString());
```



ref, out, params

- **ref**: tương tự như truyền tham chiếu trong C/C++
- Từ khoá **ref** phải được dùng lúc gọi hàm
- Các tham số truyền dạng **ref** phải được khởi tạo giá trị trước

```
public static int Main()  
{  
    int num1 = 5, num2 = 2;  
    Swap(ref num1, ref num2);  
  
    return 0;  
}  
  
public static void Swap(ref int a, ref int b)  
{  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}
```

sử dụng ref cho
tham số khi gọi
hàm

Khai báo ref trước
kiểu dữ liệu



ref, out, params

- **out**: tương tự như ref
- Khác ref là **out** ko cần khởi tạo giá trị trước khi truyền

```
public static int Main()  
{  
    int a = 10, b;  
    Subtraction(a, out b);  
  
    return 0;  
}  
  
public static void Subtraction(int x, out int y)  
{  
    y = x - 2;  
}
```

Dùng trước tham số khi gọi hàm

Khai báo cho tham số



ref, out, params

```
public static void Sum( out int result, params int[] myArray)
{
    result = 0;
    for (int i = 0; i < myArray.Length; i++)
        result += myArray[i];
}
```

Luôn khai
báo ở cuối
danh sách
tham số

```
public static int Main()
{
    int s;
    Sum(out s, 1, 2, 3);
    return 0;
}
```

3 phần tử

```
public static int Main()
{
    int s;
    int[] array = { 11, 22, 33, 44 };
    Sum(out s, array);
    return 0;
}
```

Mảng array



Keyword this

```
public class list
{
    private int size;
    ...

    public SetSize (int size)
    {
        this.size = size;
    }
}
```



LỆNH

1. Lệnh lặp for, while, do while, foreach
2. Lệnh phân nhánh switch, lệnh nhảy

Video

https://www.youtube.com/watch?time_continue=52&v=mKLRETK9slQ&feature=emb_logo



LỆNH

- Tương tự như C/C++: while, do while, for

```
while <điều kiện>
```

```
{
```

```
    // phần thân while
```

```
}
```

```
do
```

```
{
```

```
    // phần thân do while
```

```
} while <điều kiện>;
```

```
for( khởi tạo biến lặp; <điều kiện theo biến lặp>; thay đổi biến lặp)
```

```
{
```

```
    // phần thân for
```

```
}
```

Phải là giá trị bool: true, false



LỆNH

```
index = 10;  
while (index != 0){  
    Console.WriteLine(index);  
    index--;  
}
```

Giá trị {true, false}

```
index = 0;  
do{  
    Console.WriteLine("Happens at least once");  
}while (index < 0);
```

```
for(index = 0; index < 100; index++){  
    Console.Write(index);  
    Console.Write("\t");  
}
```



foreach

Cú pháp

```
foreach( typedata identifier in objectArray )  
{  
    // thân foreach  
}
```



foreach

```
public static int Main()
{
    Int32[] myArray = new Int32[] { 10, 20, 30, 40 };
    foreach (Int32 i in myArray)
    {
        Console.WriteLine(i);
    }
    return 0;
}
```

Chỉ sử dụng
biến *i* cho mỗi
lần lặp

```
public static int Main()
{
    Int32[] myArray = new Int32[] { 10, 20, 30, 40 };
    for (int i = 0; i < myArray.Length; i++)
    {
        Console.WriteLine(myArray[i]);
    }
    return 0;
}
```

Sử dụng chỉ
số mảng như
bình thường



switch

```
switch (country)
{
    case "Germany":
    case "Austria":
    case "Switzerland":
        language = "German";
        break;
    case "England":
    case "USA":
        language = "English"; break;
    case null:
        Console.WriteLine("no country specified");
        break;
    default:
        Console.WriteLine("don't know language of {0}", country);
        break;
}
```



switch

- Biểu thức switch gồm: kiểu số, ký tự, enum và chuỗi
- Sử dụng break, goto, return để điều khiển luồng thực thi
- Nếu ko nhận nào phù hợp → default
- Nếu ko có default → thực hiện lệnh sau switch



Jump

- break
 - Thoát khỏi vòng lặp
- continue
 - Qua bước lặp kế
- goto
 - Nhảy đến nhãn
 - Sử dụng goto case <expression>, trong switch

```
for(int i=0; i < myArr.Length;i++)  
{  
    if (myArr[i] <= -1)  
        break;  
    if (myArr[i] == 0)  
        continue;  
    else  
        myArr[i]++;  
}
```



return

- Thoát khỏi hàm void
- Trả về 1 giá trị của hàm

```
void Func1(int x)  
{  
  
    if (x == 0)  
  
        return;  
  
    ...  
  
}
```

```
int max(int a, int b)  
{  
  
    if (a > b)  
        return a;  
    else  
        return b;  
  
}
```




Câu hỏi

1. Sự khác nhau giữa kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu? Kiểu chuỗi, kiểu lớp trong C# là kiểu dữ liệu nào?
2. So sánh cấu trúc điều kiện if và switch
3. So sánh cấu trúc lặp while và do... while
4. Cho biết các lệnh rẽ nhánh có điều kiện và lệnh rẽ nhánh không điều kiện



Bài tập

Bài 1: Viết chương trình tính

$$S = 1 + 2 + 3 + \dots + n \text{ với } n \text{ nhập từ bàn phím}$$

Bài 2: viết chương trình tính tổng bình phương của các số sau

$$S = 1^2 + 2^2 + 3^2 + \dots + n^2 \text{ (n nhập từ bàn phím)}$$

Bài 3: Viết chương trình tính giai thừa của n (với n nhập từ bàn phím)



Bài tập

Bài 4: Viết chương trình xuất ra màn hình tam giác cân hợp bởi các ký tự ‘*’ với chiều cao =5(hình 1)

Hình 1

				*				
			*	*	*			
		*	*	*	*	*		
	*	*	*	*	*	*	*	
*	*	*	*	*	*	*	*	*

Hình 2

				*				
			*		*			
		*				*		
	*						*	
*	*	*	*	*	*	*	*	*

Bài 5: Viết chương trình xuất ra màn hình tam giác cân (rỗng) hợp bởi các ký tự ‘*’ với chiều cao nhập từ bàn phím (hình 2)



MẢNG VÀ TẬP HỢP

1. Mảng 1 chiều, đa chiều
2. Collections
3. Kiểu enumeration

Video

<https://www.youtube.com/watch?v=RcFaJxtAWDI>



MẢNG

- Chứa các biến có cùng kiểu dữ liệu.
- Truy xuất phần tử thông qua chỉ số (index)
- Chỉ số bắt đầu bằng 0.

Datatype[] array-name

- VD
 - `int[] myInteger = new int[5];`
 - `string[] myString = { "abc", "def" };`



MẢNG

- Lấy kích thước mảng qua thuộc tính Length
 - `int Size = myArray.Length;`
- Nếu thành phần của mảng là kiểu định trước, có thể dùng hàm Sort của lớp **Array** để sắp xếp
 - `Array.Sort(myArray);`
- Dùng hàm Reverse của Array để đảo thứ tự các phần tử trong mảng
 - `Array.Reverse(myArray);`



MẢNG

```
public static int Main()
{
    string[] artists = { "Leonardo", "Monet", "Van Gogh", "Klee" };

    Array.Sort(artists);

    foreach (string name in artists)
        Console.WriteLine(name);

    Array.Reverse(artists);
    foreach (string name in artists)
        Console.WriteLine(name);

    Console.ReadLine();
    return 0;
}
```

Dùng phương thức tĩnh Sort
của lớp Array để sort artists

Dùng phương thức tĩnh Reverse
của lớp Array để đảo thứ tự
artists



Datatype[,] array-name

- Khai báo mảng int 2 dòng 3 cột
 - `int[,] myMatrix = new int[2, 3];`
- Có thể khởi gán
 - `int[,] matrix1 = new int[,] { {1,2},{3,4},{5,6},{7,8}};`
 - `int[,] matrix2 = { {1,2},{3,4},{5,6},{7,8}};`
 - `string[,] beatleName = { {"Lennon","John"}, {"McCartney","Paul"}, {"Harrison","George"}, {"Starkey","Richard"}};`



MẢNG ĐA CHIỀU

```
static void Main(string[] args)
{
    double[,] matrix = new double[10, 10];
    int count = 0;
    for (int i = 0; i < matrix.GetLength(0);
        for (int j = 0; j < matrix.GetLength(1); j++)
            matrix[i, j] = ++count;

    foreach (double d in matrix)
        Console.WriteLine(d);

    Console.ReadLine();
}
```

foreach (double d in matrix)
Console.WriteLine(d);

Truy cập tuần tự theo kiểu
mảng 1 chiều

```
static void Main(string[] args)
{
    double[,] matrix = new double[10, 10];
    int count = 0;
    for (int i = 0; i < matrix.GetLength(0); i++)
        for (int j = 0; j < matrix.GetLength(1); j++)
            matrix[i, j] = ++count;

    for (int i = 0; i < matrix.GetLength(0); i++)
        for (int j = 0; j < matrix.GetLength(1); j++)
            Console.WriteLine(matrix[i, j]);

    Console.ReadLine();
}
```

for (int i = 0; i < matrix.GetLength(0); i++)
for (int j = 0; j < matrix.GetLength(1); j++)
Console.WriteLine(matrix[i, j]);

Truy cập theo dạng dòng
cột qua chỉ mục i và j



Datatype[][] array-name

- Jagged là mảng mà mỗi phần tử là một mảng có kích thước khác nhau
- Những mảng con này phải được khai báo riêng
- Khai báo mảng 3 dòng, mỗi dòng là một mảng 1 chiều

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

```
a[0] = new int[4];
```

```
a[1] = new int[3];
```

```
a[2] = new int[1];
```

→ Khai báo số dòng, hàng

→ Khai báo số cột riêng cho từng dòng



MẢNG Jagged

```
public static int Main()
{
    string[][] softwares = new string[3][];

    softwares[0] = new string[] {
        "Bitdefender", "Karperky", "NAV"};
    softwares[1] = new string[] {
        "IE", "Mozilla", "Opera", "Avant"};
    softwares[2] = new string[] {
        "MS Word", "OpenOffice"};

    for (int i = 0; i < softwares.GetLength(0); i++)
        for (int j = 0; j < softwares[i].GetLength(0); j++)
            Console.WriteLine(softwares[i][j]);

    return 0;
}
```

Truy cập theo dòng, cột

```
public static int Main()
{
    string[][] softwares = new string[3][];

    softwares[0] = new string[] {
        "Bitdefender", "Karperky", "NAV"};
    softwares[1] = new string[] {
        "IE", "Mozilla", "Opera", "Avant"};
    softwares[2] = new string[] {
        "MS Word", "OpenOffice"};

    foreach (string[] srr in softwares)
        foreach (string s in srr)
            Console.WriteLine(s);

    return 0;
}
```

Truy cập dùng foreach



TẬP HỢP

- C# hỗ trợ mạnh mẽ việc thao tác trên tập hợp
- Collection là enumerable data structures thông qua index hoặc key.
- Namespace
 - System.Array
 - System.Collections
- System.Collections cung cấp các lớp, phương thức, thuộc tính để tương tác với nhiều cấu trúc dữ liệu khác nhau.



TẬP HỢP

- Các giao diện được định nghĩa trong namespace

IEnumerable

IEnumerator

ICollection

ICollection

IDictionary



TẬP HỢP

- Các lớp thực thi ICollection
 - System.Collections.Stack
 - System.Collections.Queue
 - System.Collections.BitArray
 - System.Collections.NameValueCollection



TẬP HỢP

- IDictionary thể hiện các collection theo dạng name-value.
- Các lớp thực thi giao diện này
 - System.Collections.SortedList
 - System.Collections.Hashtable
 - System.Collections.Specialized.HybridDictionary
 - System.Collections.Specialized.ListDictionary



TẬP HỢP

- IList thể hiện collection chỉ có giá trị.
- Những lớp thực thi giao diện này
 - System.Array
 - System.Collections.ArrayList
 - System.Collections.Specialized.StringCollection



TẬP HỢP

- Giao diện IEnumerable

- Cung cấp khả năng duyệt forward, read-only các item của đối tượng.
- Phương thức chính GetEnumerator() trả về đối tượng thực thi giao diện IEnumerator.

```
string[] name = { "Ha Giang", "Ha Nam", "Ha Noi", "Ha  
Tay" };  
for(IEnumerator e = name.GetEnumerator(); e.MoveNext();  
    Console.WriteLine(e.Current));
```

```
string[] name = { "Ha Giang", "Ha Nam", "Ha Noi", "Ha  
Tay" };  
foreach(string s in name)  
    Console.WriteLine(s);
```



TẬP HỢP

- ArrayList: mảng động những đối tượng không cùng kiểu dữ liệu.
- Array chỉ chứa các đối tượng cùng kiểu dữ liệu
- ArrayList có khả năng chứa các đối tượng khác kiểu dữ liệu, nhưng được chứa dưới dạng kiểu Object
- VD: một đối tượng ArrayList có thể chứa các item kiểu string, int, long, float...
- ArrayList sử dụng indexer để xác định các item



TẬP HỢP

- Thuộc tính Count cho biết số lượng item được lưu trữ trong collection.
- Thuộc tính Capacity cho phép get/set số lượng item mà ArrayList có thể lưu trữ.
- Các item được thêm vào qua phương thức Add và xóa qua Remove



- Minh họa ArrayList

```
int i = 100;  
float f = 99.99f;  
ArrayList arraylist = new ArrayList();  
arraylist.Add("Ha Nam");  
arraylist.Add(i);  
arraylist.Add(f);  
for (int index = 0; index < arraylist.Count;  
index++)  
    Console.WriteLine(arraylist[index]);
```



TẬP HỢP

- StringCollection thực thi giao diện IList và tương tự như ArrayList các String

```
StringCollection strlist = new  
StringCollection();  
strlist.Add("Ha Giang");  
strlist.Add("Ha Nam");  
strlist.Add("Ha Noi");  
foreach(string str in strlist)  
    Console.WriteLine(str);
```



TẬP HỢP

- StringDictionary là dạng hashtable có khóa và dữ liệu cùng dạng string
- Hashtable chứa dữ liệu trong các khóa của nó

```
StringDictionary strList = new  
StringDictionary();  
strList.Add("A", "Ha Nam");  
strList.Add("B", "Ha Giang");  
strList.Add("C", "Ha Noi");  
  
foreach(string s in strList.Values) // duyệt qua  
từng giá trị  
    Console.WriteLine(s);
```



TẬP HỢP

- Duyệt qua từng item trong StringDictionary
- Sử dụng kiểu DictionaryEntry

```
StringDictionary strList = new  
StringDictionary();  
strList.Add("A", "Ha Nam");  
strList.Add("B", "Ha Giang");  
strList.Add("C", "Ha Noi");  
  
foreach(DictionaryEntry d in strList)  
    Console.WriteLine("Key: {0} ==> Value:  
{1}", d.Key, d.Value);
```



TẬP HỢP

- Stack cung cấp danh sách dạng LIFO cho các item có kiểu Object.
- Các thao tác cơ bản Push và Pop

```
Stack objs = new Stack();  
objs.Push("Ha Giang");  
objs.Push("Ha Nam");  
objs.Push("Ha Noi");  
While (objs.Count > 0)  
    Console.WriteLine(objs.Pop());
```





TẬP HỢP

- Queue: cung cấp danh sách dạng FIFO
- Item được thêm vào cuối danh sách và lấy ra ở đầu danh sách

```
Queue objs = new Queue();
```

```
objs.Enqueue("Ha Giang");
```

```
objs.Enqueue("Ha Nam");
```

```
objs.Enqueue("Ha Noi");
```

```
while (objs.Count > 0)
```

```
    Console.WriteLine(objs.Dequeue());
```

Đây là người xếp hàng đầu tiên khi đến mua vé.
Chắc chắn người này sẽ mua được vé đầu tiên.





TẬP HỢP

- Hashtable: cung cấp cách thức nhanh chóng để lưu trữ và truy cập những item có kiểu object.
- Hỗ trợ tìm kiếm theo khóa

```
Hashtable hashtable = new Hashtable();
```

```
hashtable.Add(1, "Ha Nam");
```

```
hashtable.Add(2, "Ha Giang");
```

```
hashtable.Add(3, "Ha Noi");
```

```
foreach(int key in hashtable.Keys) //xuất các  
khóa
```

```
    Console.WriteLine(key);
```



- Sử dụng IDictionaryEnumerator

```
Hashtable hashtable = new Hashtable();  
hashtable.Add(1, "Ha Nam");  
hashtable.Add(2, "Ha Giang");  
hashtable.Add(3, "Ha Noi");
```

```
IDictionaryEnumerator en =  
hashtable.GetEnumerator();  
while (en.MoveNext())  
    Console.WriteLine("Key: {0} ==> Value:  
{1}",  
                        en.Key, en.Value);
```



TẬP HỢP

- SortedList: cho phép lưu trữ các item theo dạng khóa-giá trị, hỗ trợ sắp xếp các item

```
SortedList slist = new SortedList();  
slist.Add(3, "Ha Giang");  
slist.Add(1, "Ha Nam");  
slist.Add(2, "Ha Noi");
```

```
foreach (string s in slist.Values) // duyệt qua  
các giá trị  
    Console.WriteLine(s);
```



TẬP HỢP

- Hỗ trợ collection với kiểu dữ liệu bất kỳ (điểm mới từ .NET 2.0)
- Các tính năng nổi bật
 - Type safe
 - No boxing/unboxing
 - Richer functionality through System defined delegate types



Generic Collections

- List<T>

```
List<int> list = new List<int>();
```

```
list.Add(100); // thêm 1 phần tử
```

```
list.AddRange(new int[] {33,44,55}); // thêm nhiều phần tử
```

```
list.Insert(list.Count, 9999); // chèn vào cuối
```

```
foreach(int i in list)  
    Console.WriteLine(i);
```

Phân biệt List<T> và ArrayList ?



Generic Collections

- Dictionary<T>

```
struct Car {  
    public string model;  
    public Car(string m) {  
        model = m;  
    }  
}
```

```
Dictionary<int, Car> showroom = new Dictionary<int,  
Car>();
```

```
showroom.Add(1, new Car("Camry"));
```

```
showroom.Add(200, new Car("Lexus"));
```

```
showroom.Add(3000, new Car("Accura"));
```

```
foreach(Car c in showroom.Values)
```

```
    Console.WriteLine(c.model);
```



Enumeration

- Dùng thay thế hằng
- Tập hợp các giá trị hằng được đặt tên
- Khai báo trực tiếp trong namespace
- Là kiểu dữ liệu

```
enum Color { Red, Green, Blue };  
enum Access { personal = 1, group = 2, all = 4 };
```

Sử dụng

```
Color c = Color.Red;
```

```
Access a = Access.personal | Access.group;  
If ((Access.personal & a) != 0)  
    Console.WriteLine("access granted");
```




Enumeration

- Enumeration kế thừa từ object (Equals, ToString()).

Compare

+, -

++, --

&

|

~

if (c == Color.red) ...

if (c > Color.red && c <= Color.green) ...

c = c + 2;

c++;

if ((c & Color.red) == 0) ...

c = c | Color.blue;

c = ~ Color.red;



Enumeration

```
public enum TimeOfDay
{
    Morning = 0,
    Afternoon = 1,
    Evening = 2
}

public static int Main()
{
    TimeOfDay day;
    day = TimeOfDay.Evening;
    WriteGreeting(day);
    return 0;
}

static void WriteGreeting(TimeOfDay timeOfDay)
{
    switch (timeOfDay)
    {
        case TimeOfDay.Morning:
            Console.WriteLine("Good morning!");
            break;
        case TimeOfDay.Afternoon:
            Console.WriteLine("Good afternoon!");
            break;
        case TimeOfDay.Evening:
            Console.WriteLine("Good evening!");
            break;
        default:
            Console.WriteLine("Hello!");
            break;
    }
}
```



Tóm tắt

- Cú pháp khá giống với C/C++
- Kiểu dữ liệu tham chiếu & giá trị
- Truyền tham số kiểu giá trị cho hàm
- Type-cast
- Boxing & Unboxing
- Điều khiển lặp foreach duyệt tập hợp
- Kiểu dữ liệu mảng
- Collections
- Enumeration



Bài tập

- Viết chương trình giải phương trình bậc 2



Bài tập

- Viết chương trình nhập vào một tháng, tính số ngày của tháng đó
 - Khi người dùng nhập vào số <1 hoặc >12 , báo không tồn tại tháng.
 - Khi người dùng nhập vào tháng 1, 3, 5, 7, 8, 10, 12 thông báo “tháng X có 31 ngày”.
 - Khi người dùng nhập vào tháng 4, 6, 9, 11 thông báo “tháng X có 30 ngày”
 - Khi người dùng nhập vào tháng 2, máy tính sẽ hỏi năm nào nếu năm nhuận thì có 29 ngày.
 - *Lưu ý*: Năm nhuận – năm không chia hết cho 100 nhưng lại chia hết cho 4, hoặc là năm chia hết cho 400 thì tháng 2 có 29 ngày. Những năm khác không nhuận, tháng 2 có 28 ngày.



Bài tập

- Viết chương trình Nhập họ tên, ngày sinh giới tính của một nhân viên
- Tính ngày về hưu của nhân viên đó biết:
 - Nếu giới tính là nam thì 60 tuổi
 - Nếu giới tính là nữ thì 55 tuổi
- Nếu ngày sinh chính là ngày hôm nay thì ghi ra là chúc mừng sinh nhật



Bài tập

- Viết chương trình Nhập một mảng các số nguyên
- Tính tổng các số nguyên của mảng đó
 - Số lớn nhất trong mảng đó là số nào
 - Số lượng và tổng các số nguyên dương của mảng đó



Bài tập

- Viết chương trình tạo một Dictionary<string, string> với khóa là mã môn học, giá trị value là tên của các môn học trong học kỳ 1 năm 2019-2020 của từng sinh viên.
- Viết lệnh nhập tất cả các môn học vào trong Dictionary
- Thực hiện kiểm tra xem môn “Kỹ thuật thương mại điện tử” đã tồn tại hay chưa, nếu chưa thì chèn môn “Kỹ thuật thương mại điện tử” vào với khóa là IS311.
- Đếm xem có bao nhiêu môn học
- Thực hiện xóa môn CS414 nếu có.



Tài liệu tham khảo

- [1] Giáo trình lập trình Winform với C#.NET Lê Trung Hiếu, Nguyễn Thị Minh Thi
- [2] Giáo trình lập trình C#.net Phạm Hữu Khang
- [3] C# Language Reference, Anders Hejlsberg and Scott Wiltamuth, Microsoft Corp.
- [4] Professional C#, 2nd Edition, Wrox Press Ltd.
- [5] Web site www.Codeproject.com
- [6] Web site www.CodeGuru.com