

ASP.NET Core



Lương Trần Hy Hiền
hyhien@gmail.com
0989.366.990



Thông tin

- ❑ Web: **<https://hienlth.info/aspcore>**
 - Giáo trình, bài tập – online
 - Dự phòng: **<https://aspcore.weebly.com/>**
- ❑ ~24 buổi, tuần 2 buổi, 4h/buổi (T7CN)
- ❑ ~32 buổi, tuần 3 buổi, 3h/buổi (246/357)
- ❑ Liên hệ: Lương Trần Hy Hiến
0989.366.990
hyhien@gmail.com

C# Recommended Software

❑ Software needed for this course:

- Microsoft Windows 10
- Visual Studio 2019 – <https://www.visualstudio.com>
- MS SQL Server - <https://www.microsoft.com/en-cy/sql-server/sql-server-downloads>



Microsoft®
SQL Server®



Visual Studio



.NET Core Platform

Something New, Something Old

.NET Core Platform

☐ Cross-platform

- .NET Core apps that run on Windows, Linux and macOS

☐ Flexible

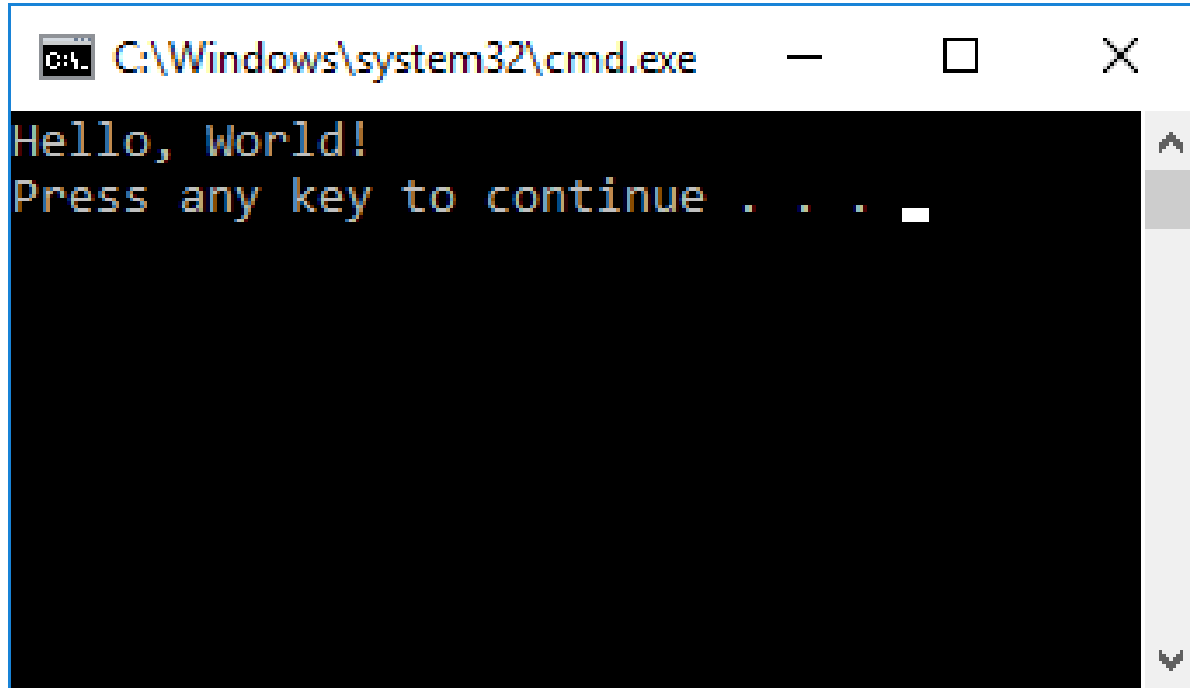
- Each component is a package
- Metapackages

☐ Lightweight

- Projects include only required dependencies

☐ Fully open-source at [GitHub](https://github.com/dotnet/core)

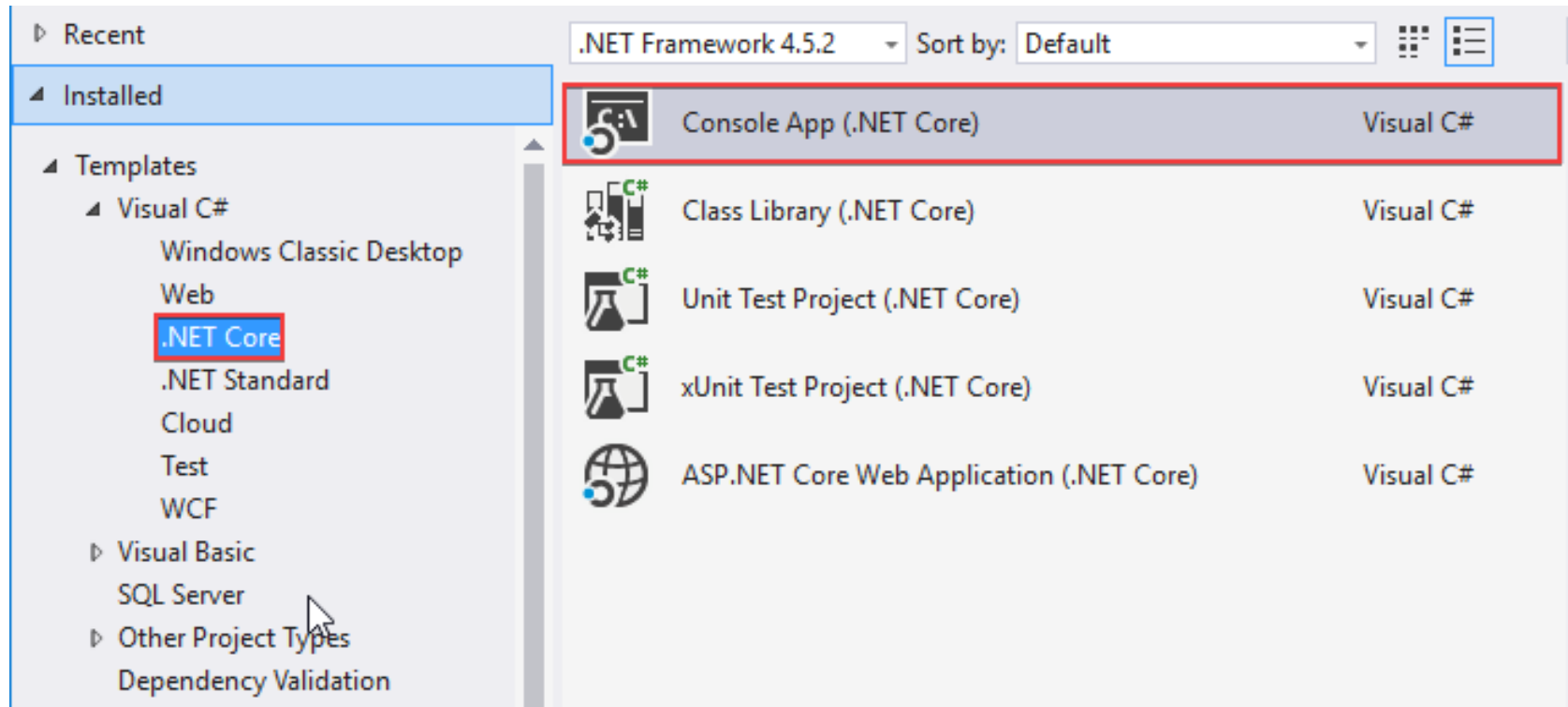
Simple .NET Core Console Apps

A screenshot of a Windows Command Prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe' and standard window controls. The command prompt has a black background with white text. It displays 'Hello, World!' on the first line and 'Press any key to continue' on the second line, followed by a white cursor. A vertical scrollbar is visible on the right side of the window.

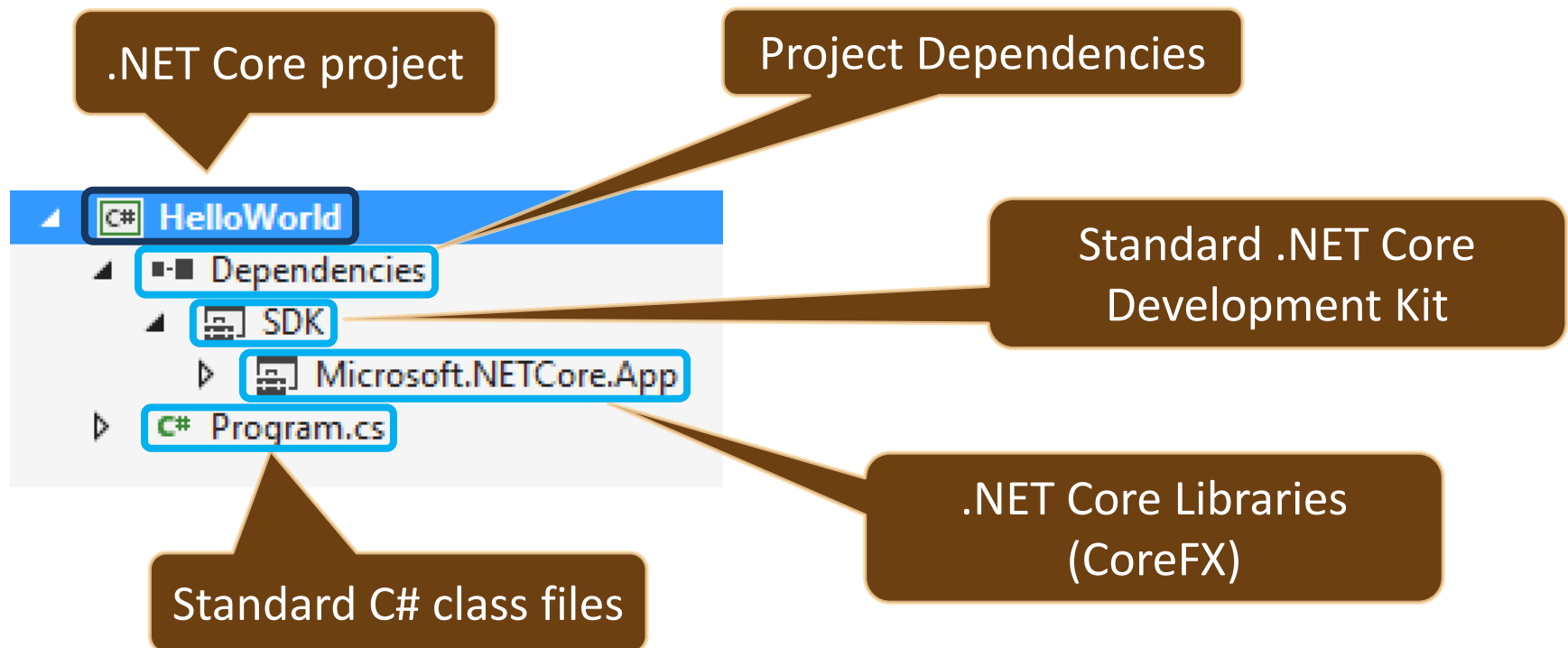
```
C:\Windows\system32\cmd.exe  
Hello, World!  
Press any key to continue . . . .
```

Hello, .NET Core!

Creating console application



Project Structure



Hello World Code

.NET Core

```
using System;
```

```
namespace HelloWorld
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

.NET Framework

```
using System;
```

```
namespace HelloWorld
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Khái niệm cơ bản C#

Nội dung

- ❑ Giới thiệu C#
- ❑ Môi trường lập trình
- ❑ Kiểu dữ liệu
- ❑ Khai báo biến
- ❑ Rẻ nhánh, Lặp
- ❑ Hàm, mảng, Tập hợp

Các loại ứng dụng C#

□Chương trình Console (Console App)

- ▣ Giao tiếp với người dùng bằng bàn phím
- ▣ Không có giao diện đồ họa (GUI)

□Chương trình Class Library

- ▣ Định nghĩa các thư viện cho các ứng dụng khác dùng

□Chương trình Web App/MVC

- ▣ Web Page (Razor Page)
- ▣ Mô hình MVC

Từ khoá C#

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit

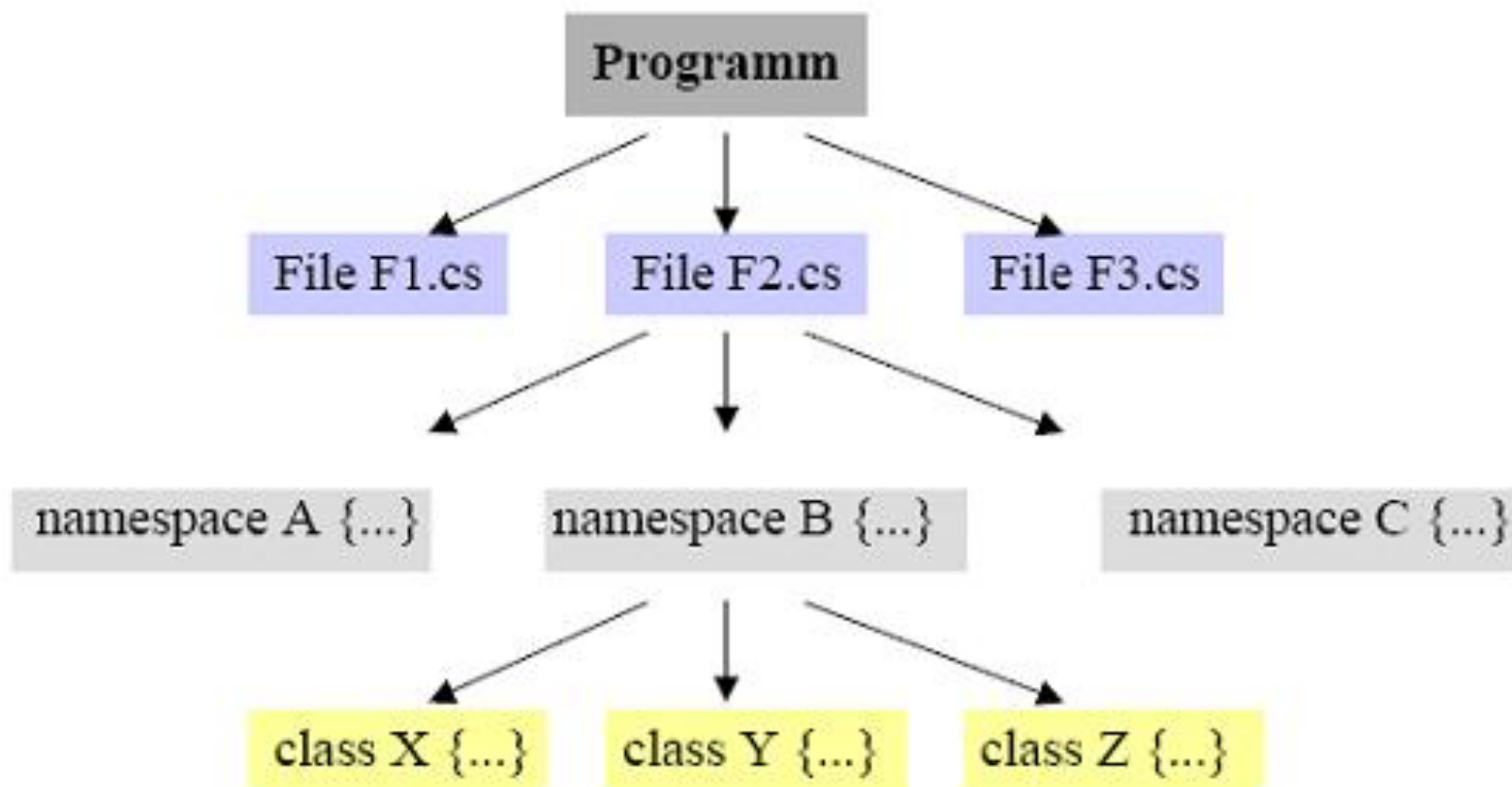
Từ khoá C#

in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short

Từ khoá C#

sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	using static	virtual	void
volatile	while		

Cấu trúc 1 chương trình C#



Những cơ sở ngôn ngữ C#

□ Phân biệt chữ hoa chữ thường

□ Có các kiểu :

- Dựng sẵn: **byte**, **char**, **string**, **int**, **float**, **double**, **bool**,

...

- Hằng : `const int PI = 3.1416;`

- Liệt kê :

enum Ngay

{

Hai, Ba, Tu, Nam, Sau, Bay, ChuNhat

};

KIỂU NGÀY GIỜ

- ▶ **DateTime**: lưu trữ thời gian (Year, Month, Date, Hour, Minute, Second, Millisecond)
- ▶ **DateTime.Now**: thời gian hiện tại
- ▶ **DateTime.Parse**(chuỗi): chuỗi phải có dạng M/d/yyyy
- ▶ **DateTime.ToString**(định dạng)
- ▶ Định dạng:
 - Năm: yy, yyyy
 - Tháng: M, MM, MMM, MMMM
 - Ngày: d, dd
 - Giờ: H, HH, h, hh
 - Phút: m, mm
 - Giây: s, ss
 - Sáng/chiều: a

```
DateTime d1 =  
DateTime.Parse("12/31/2000");
```

```
DateTime d2 = DateTime.Now;  
String s = d2.ToString("H/MM");  
Response.Write(s);
```

Khai báo biến, hằng

□ Khai báo biến:

< *kiểu DL* > < *Tên_biến* > [= < *giá_trị* >];

□ Ví dụ:

int i;

i = 0;

int x = 10, y = 20, z = 10;

bool b = true;

□ Khai báo hằng:

const < *kiểu DL* > < *Tên_hằng* > = < *giá_trị* >;

const int LUONGCB = 1050;

Xuất dữ liệu

❑ Nhập dữ liệu từ bàn phím và xuất dữ liệu ra màn hình trong C# có thể dùng các **phương thức tĩnh** trong lớp: System.**Console**

- Xuất dữ liệu lên màn hình

```
void Console.Write(data) ;  
void Console.WriteLine(data) ;
```

Nhập dữ liệu

❑ Nhập dữ liệu từ bàn phím

- Cú pháp:

```
int Console.Read() ;  
string Console.ReadLine() ;
```

- Trong đó: `Console.Read()` trả về mã ASCII của ký tự đầu tiên.

Ví dụ nhập/xuất

□ Ví dụ:

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

```
Console.WriteLine(s);
```

```
int a = 10;
```

```
int b = 20;
```

```
int kq = a + b;
```

```
Console.WriteLine($"{a} + {b} = {kq}.");
```

\$ - string interpolation

- ❑ Giúp format các biến dễ dàng không qua các index.
- ❑ Tương tự như `string.Format` trước đây.
- ❑ Ví dụ:

```
int[] values = { 10, 20, 30 };
```

```
int cats = 2, dogs = 4;
```

```
string animals = $"cats = {cats} and dogs = {dogs}";
```

```
string result1 = $"The second value is {values[1]}";
```

```
string result2 = $"The paw count is {Paws(5)}";
```

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/interpolated>

Ví dụ nhập/xuất(tt)

□ Ví dụ: nhập số nguyên và số thực

```
int n;
```

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

```
n = int.Parse(s);
```

```
double f;
```

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

```
f = double.Parse(s);
```


Nhập dữ liệu – Chuyển kiểu dữ liệu

- ❑ Để chuyển một kiểu dữ liệu sang một kiểu dữ liệu khác chúng ta dùng cú pháp sau
- ❑ Cú pháp

```
Kieu.Parse("chuo") ;
```

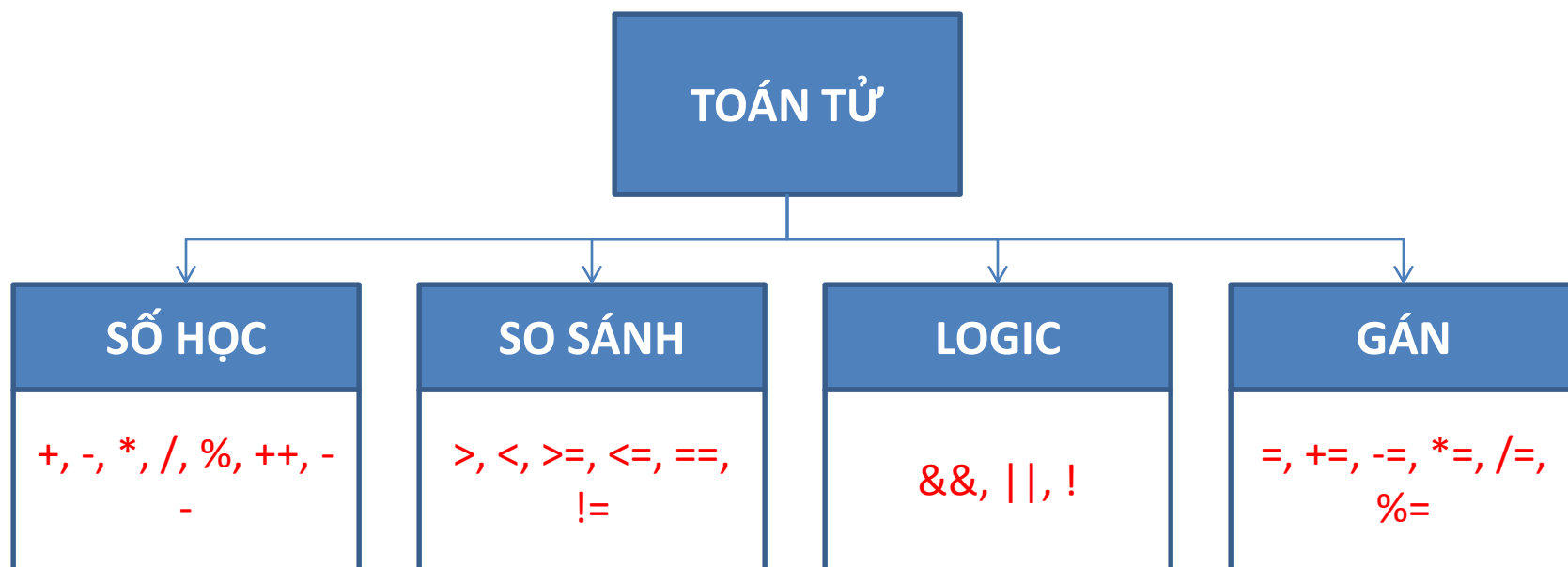
- Ví dụ:

```
string s = "123" ;  
int data = int.Parse(s) ;
```

Nhập dữ liệu – Lớp Convert

Phương thức	Ý nghĩa
ToBoolean	Chuyển một giá trị sang giá trị Boolean
ToByte	Chuyển một giá trị sang giá trị số nguyên 8-bit không dấu
ToChar	Chuyển một giá trị sang giá trị ký tự unicode
ToDateTime	Chuyển một giá trị sang giá trị DateTime.
ToDecimal	Chuyển một giá trị sang giá trị Decimal.
ToDouble	Chuyển một giá trị sang giá trị số thực có độ chính xác gấp đôi 8 byte
ToInt16	Chuyển một giá trị sang giá trị số nguyên 16-bit có dấu
ToInt32	Chuyển một giá trị sang giá trị số nguyên 32-bit có dấu
ToInt64	Chuyển một giá trị sang giá trị số nguyên 64-bit có dấu
ToSByte	Chuyển một giá trị sang giá trị số nguyên 8-bit có dấu
ToSingle	Chuyển một giá trị sang giá trị số thực có độ chính xác đơn
ToString	Chuyển một giá trị sang giá trị một chuỗi
ToUInt16	Chuyển một giá trị sang giá trị số nguyên 16-bit không dấu
ToUInt32	Chuyển một giá trị sang giá trị số nguyên 32-bit không dấu
ToUInt64	Chuyển một giá trị sang giá trị số nguyên 64-bit không dấu

TOÁN TỬ VÀ BIỂU THỨC



Biểu thức là sự kết hợp giữa toán tử và toán hạng.
Kết quả của biểu thức cho chúng ta một giá trị. Hãy xét ví dụ sau

- `int x = 11 % 4; // giá trị của x là 3`
- `bool a = 9 < 2 + 5 && true || 4 > 3; // giá trị của y là false`

TOÁN TỬ SỐ HỌC

Phép toán	Mô tả
+	Tính tổng của 2 số
-	Tính hiệu của 2 số
*	Tính tích của 2 số
/	Tích thương của 2 số
%	Thực hiện chia có dư của 2 số
++	Tăng giá trị của biến lên 1 đơn vị
--	Giảm giá trị của biến xuống 1 đơn vị

TOÁN TỬ SO SÁNH

Toán Tử	Mô tả
==	Dùng để kiểm tra điều kiện bằng
>	Dùng để kiểm tra điều kiện lớn hơn
>=	Dùng để kiểm tra điều kiện lớn hơn hoặc bằng
<	Dùng để kiểm tra điều kiện nhỏ hơn
<=	Dùng để kiểm tra điều kiện nhỏ hơn hoặc bằng
!=	Dùng để kiểm tra điều kiện khác
<>	Dùng để kiểm tra điều kiện khác

TOÁN TỬ LOGIC

Toán Tử	Mô tả
&&	Trả về giá trị true khi tất cả biểu thức tham gia biểu thức có giá trị true
	Trả về giá trị true khi có 1 biểu thức tham gia biểu thức có giá trị là true
!	Lấy giá trị phủ định của biểu thức

TOÁN TỬ ĐIỀU KIỆN

- Toán tử điều kiện là toán tử 3 ngôi duy nhất, nó dùng để **rút gọn cách viết if-else** trong tình huống đơn giản.
- Cú pháp

<điều kiện> ? <giá trị đúng> : <giá trị sai>

Ví dụ: tìm số lớn nhất của 2 số a và b

```
int a = 1, b = 9;
```

```
int max = a > b ? a : b;
```

Một số thao tác trên chuỗi

<biến chuỗi>.ToLower();
<biến chuỗi>.ToUpper();
<biến chuỗi>.Substring(vị trí, số ký tự);
<biến chuỗi>.Length ; //không có (và)
<biến chuỗi>[vị trí]

□ Ví dụ

```
string S = "hello woRld";  
string u = S.ToUpper();  
char c = S[1]; // c = 'e'  
int l = S.Substring(0,4).Length ;  
//thay vì ghi (S.Substring(0,4)).Length
```


Một số thao tác trên chuỗi

- ❑ `IndexOf()`, `IndexOfAny()`, `LastIndexOf()`, `LastIndexOfAny()`: tìm kiếm chuỗi ký tự, hoặc một phần chuỗi ký tự trong một xâu cho trước.
- ❑ `Replace()`: thay thế một mẫu trong xâu bởi một chuỗi ký tự khác.
- ❑ `Split()`: cắt một xâu thành các xâu con dựa theo ký tự phân cách cho trước.
- ❑ `Trim()`, `TrimEnd()`, `TrimStart()`: xóa các ký tự trắng ở đầu, cuối xâu.
- ❑ `Insert()`, `Remove()`: chèn vào, xóa đi một xâu con trong một xâu cho trước.
- ❑ `StartsWith()`, `EndsWith()`: kiểm tra xem xâu có bắt đầu, kết thúc bởi một xâu khác.

Một số phương thức toán học

- ❑ `Math.Abs`(biểu thức số)
- ❑ `Math.Sqrt`(biểu thức số)
- ❑ `Math.Ceiling`(biểu thức số)
- ❑ `Math.Floor`(biểu thức số)
- ❑ `Math.Max`(biểu thức số)
- ❑ `Math.Min`(biểu thức số)
- ❑ `Math.Round`(biểu thức số)
- ❑ hằng số `Math.PI` và `Math.E`

Một số phương thức toán học

□ Ví dụ

```
float R=12.6;
```

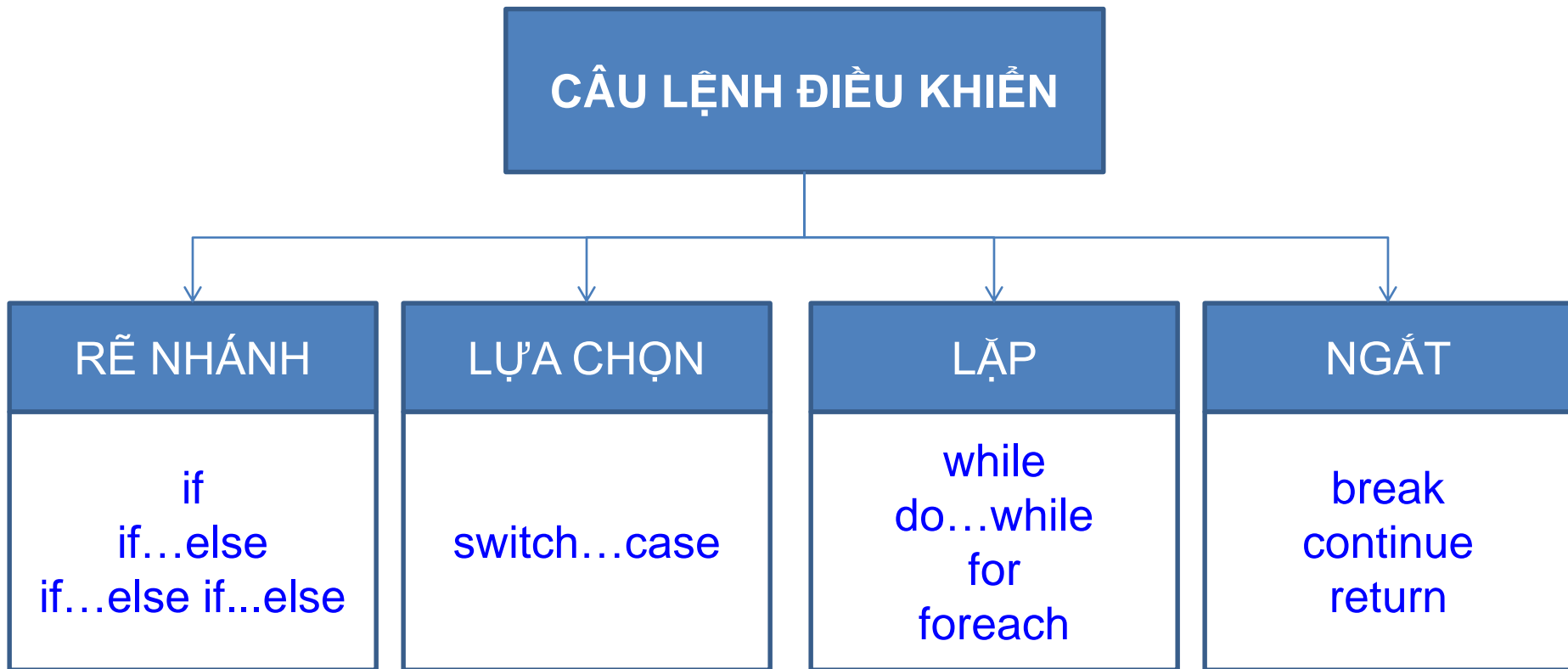
```
float S = R*R*Math.PI;
```

```
int a,b,c,d;
```

```
//Nhập 4 số a,b,c và d
```

```
Console.Write(Math.Max(a,Math.Max(b,Math.Max(c,d))));
```

CÂU LỆNH ĐIỀU KHIỂN



CÂU LỆNH ĐIỀU KHIỂN

```
graph TD; A[CÂU LỆNH ĐIỀU KHIỂN] --> B[RỄ NHÁNH<br/>(if, if...else, if...else if....else)]; A --> C[LỰA CHỌN<br/>(switch...case, break)]; A --> D[LẶP<br/>(for, while, do...while, foreach,<br/>break, continue)];
```

RỄ NHÁNH
(if, if...else, if...else if....else)

LỰA CHỌN
(switch...case, break)

LẶP
(for, while, do...while, foreach,
break, continue)

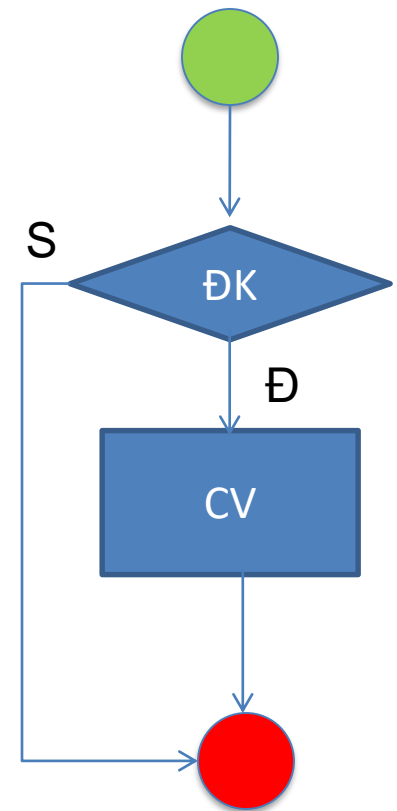
RỄ NHÁNH-IF

Cấu trúc của câu lệnh if

```
if(điều kiện)
{
    // Thực hiện khi điều kiện là true
}
```

Ví dụ:

```
int a = 1;
if(a>1)
{
    Console.WriteLine("a lon hon 1");
}
```



RỄ NHÁNH-IF...ELSE

Cấu trúc của câu lệnh điều kiện if-else:

```
if(điều kiện)
```

```
{
```

```
    // Thực hiện khi điều kiện là true
```

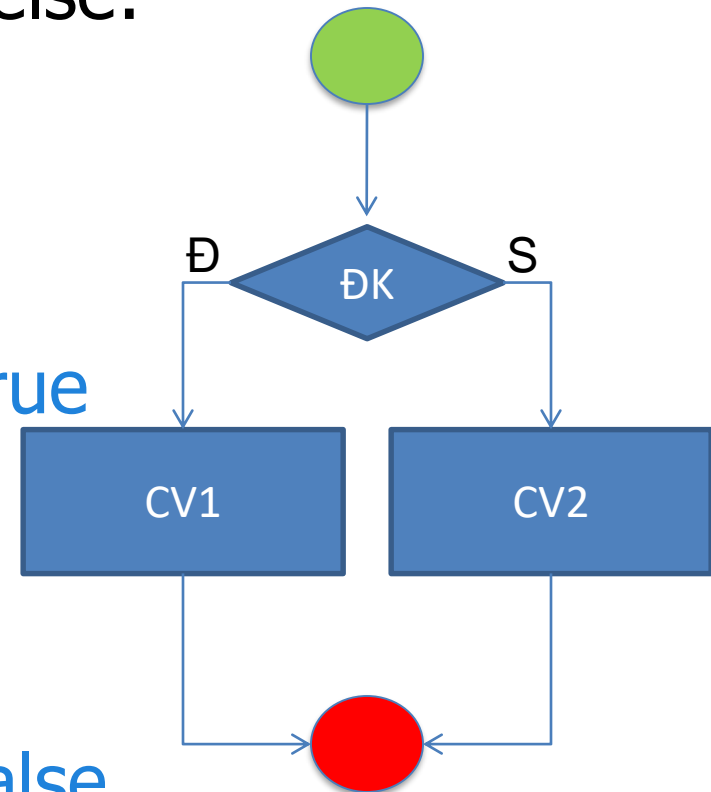
```
}
```

```
else
```

```
{
```

```
    // Thực hiện khi điều kiện là false
```

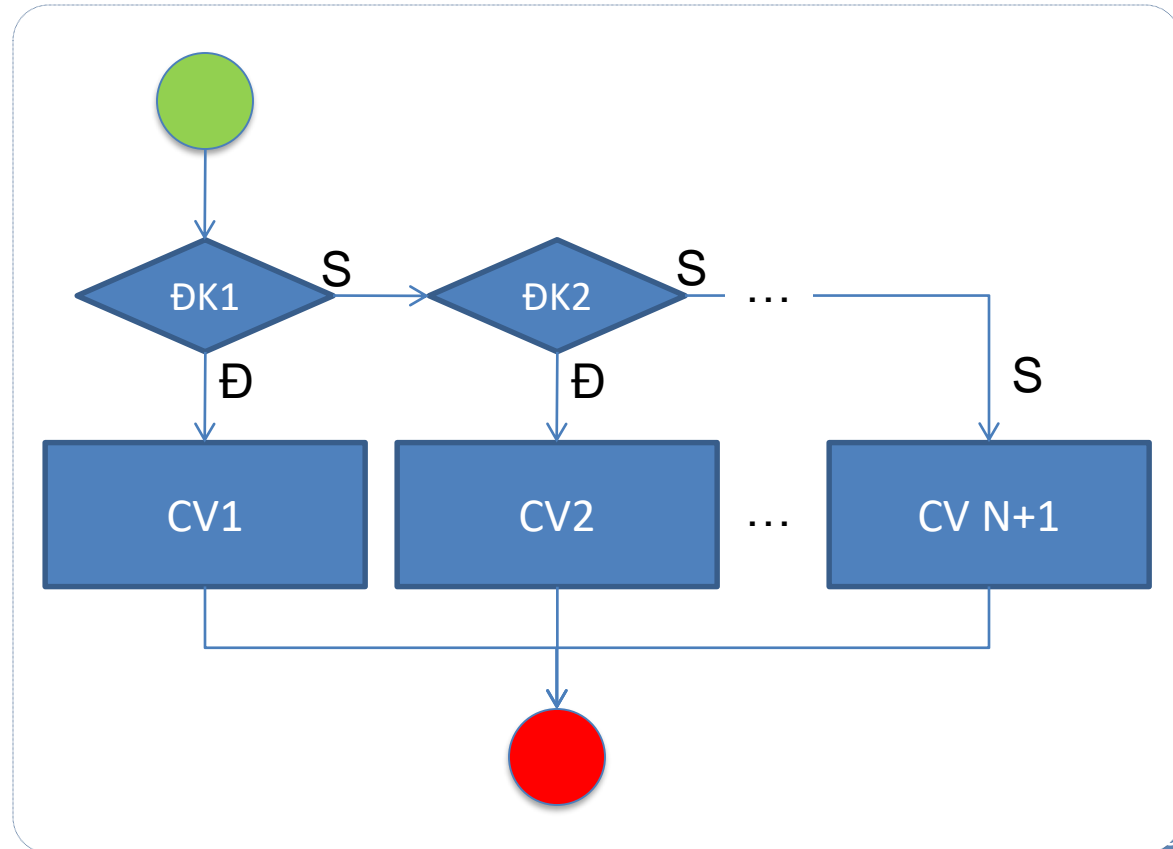
```
}
```



RỄ NHÁNH - IF...ELSE IF

- Câu lệnh điều kiện if-else if dùng để lựa chọn nhiều điều kiện.
- Cấu trúc của câu lệnh điều kiện if-else if:

```
if(điều kiện 1)
{
    // Công việc 1
}
else if(điều kiện 2)
{
    // Công việc 2
}
....
else
{
    // Công việc N+1
}
```



VÍ DỤ: IF...ELSE IF...ELSE

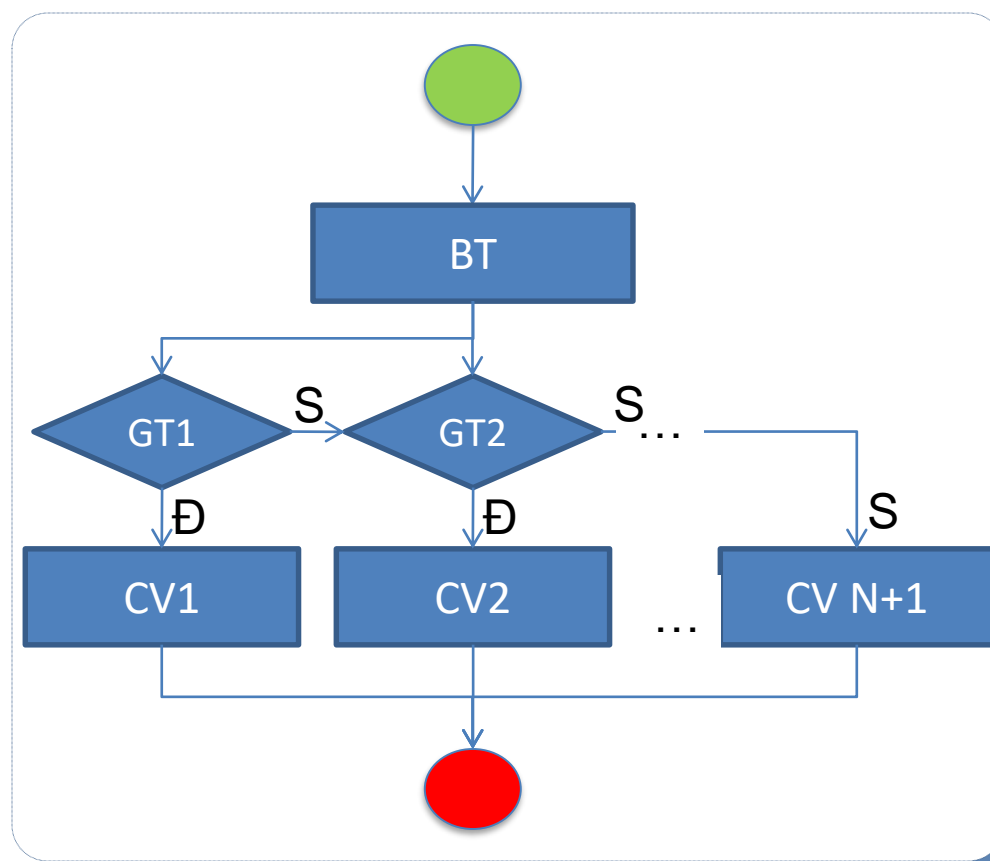
```
int hour = DateTime.Now.Hour;
if (hour < 12)
{
    Console.WriteLine("Good morning!");
}
else if (hour < 17)
{
    Console.WriteLine("Good afternoon!");
}
else
{
    Console.WriteLine("Good evening!");
}
```

LỰA CHỌN-SWITCH...CASE

- Dùng để thay thế câu lệnh điều kiện if-else if trong trường hợp có nhiều lựa chọn
- Cấu trúc của câu lệnh switch:

switch(biểu thức số học)

```
{
    case [giá trị 1]:
        // CV 1
        break;
    case [giá trị 2]:
        // CV 2
        break;
    default:
        // CV N+1
        break;
}
```



VÍ DỤ: Switch...case

```
String tt = Console.ReadLine();  
String a = Console.ReadLine();  
String b = Console.ReadLine();  
  
switch(tt[0])  
{  
    case '+':  
        int kq = int.Parse(a) + int.Parse(b);  
        String thongBao= "TONG: " + kq;  
        break;  
    case '-':  
        int kq = int.Parse(a) - int.Parse(b);  
        String thongBao= "HIEU: " + kq;  
        break;  
    default:  
        String thongBao = "KHONG HIEU";  
        break;
```

VÍ DỤ (2): SWITCH...CASE

```
DateTime valentine = DateTime.ParseExact("14-2-2011", "d-M-yyyy", null);
DayOfWeek weekDay = valentine.DayOfWeek;
switch (weekDay)
{
    case DayOfWeek.Monday:
        Console.WriteLine("Ngày đầu tuần");
        break;
    case DayOfWeek.Saturday:
    case DayOfWeek.Sunday:
        Console.WriteLine("Ngày cuối tuần");
        break;
    default:
        Console.WriteLine("Ngày trong tuần");
        break;
}
```

Kiểu liệt kê (enum)

Không break->không lệnh

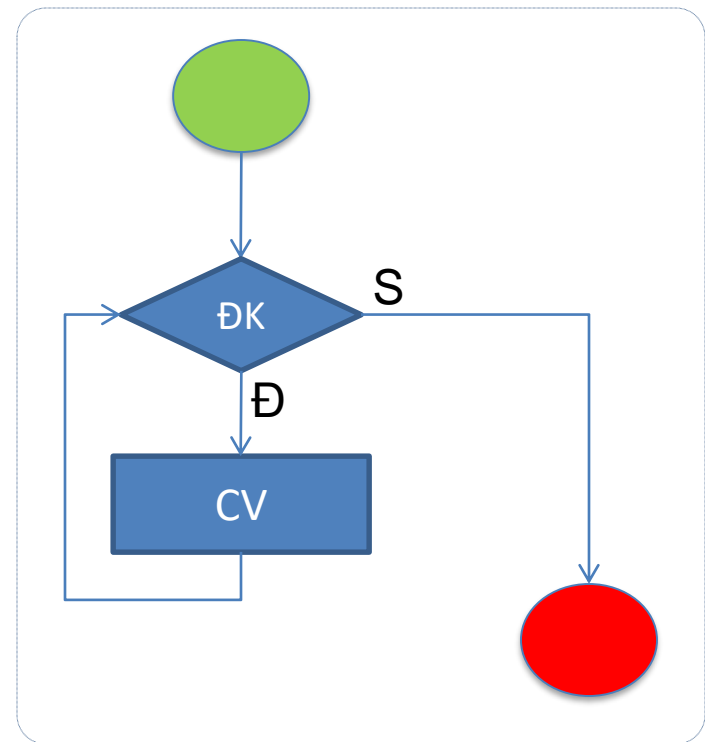
LẶP - WHILE

- Mục đích của vòng lặp while dùng để thực hiện các hành động có tính chất lặp đi lặp lại.
- Cấu trúc của vòng lặp while:

```
while(điều kiện)
{
    // Thực hiện với mỗi lần lặp
}
```

Ví dụ:

```
int i = 1;
while(i < 20)
{
    Console.WriteLine("Hello World ");
    i++;
}
```



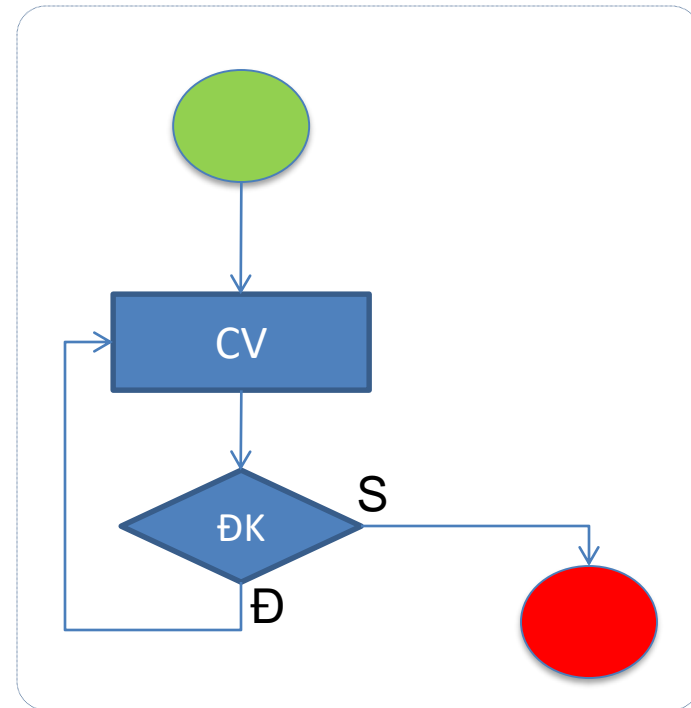
LẶP – DO...WHILE

- Mục đích của vòng lặp do-while dùng để thực hiện các hành động có tính chất lặp đi lặp lại.
- Cấu trúc của câu lệnh do-while:

```
do {  
    // Thực hiện với mỗi lần lặp  
}  
while(điều kiện);
```

Ví dụ:

```
do  
{  
    Console.WriteLine("Hello World");  
    i++;  
}  
while(i < 20);
```



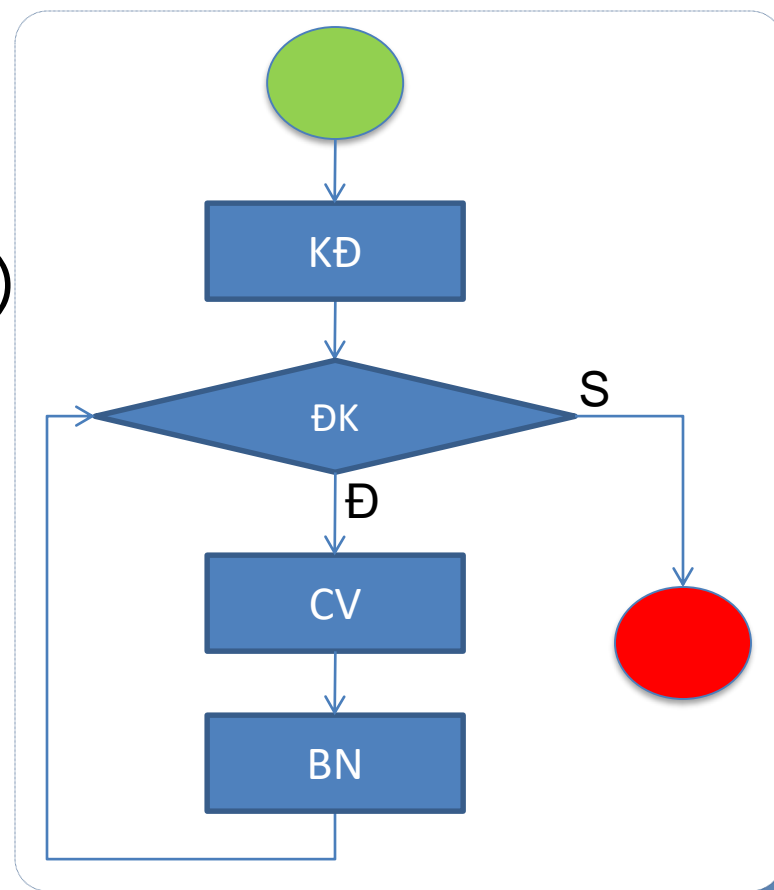
LẶP-FOR

- Mục đích của vòng lặp for dùng để thực hiện các hành động có tính chất lặp đi lặp lại.
- Cấu trúc của câu lệnh for:

```
for(khởi đầu ; điều kiện; bước nhảy)
{
    // Thực hiện với mỗi lần lặp
}
```

Ví dụ:

```
for(int i = 1; i < 20 ; i++)
{
    Console.WriteLine("Hello World ");
}
```



LẶP-BREAK

- Mục đích của câu lệnh break dùng để ngắt ra khỏi vòng lặp.
- Những câu lệnh sau break sẽ không được thực hiện.

Ví dụ:

```
for(int i=1; i < 20; i++)  
{  
    Console.WriteLine("Hello World " + i);  
    if(i > 5)  
    {  
        break;  
    }  
}
```



Các lệnh sau lặp

LẶP - CONTINUE

- Mục đích của câu lệnh continue là tiếp tục thực hiện các điều kiện tiếp theo trong vòng lặp

Ví dụ:

```
for(int i=1; i<20; i++)  
{  
    if(i % 2 != 0)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```



Thực hiện lần
lặp tiếp theo

Bài tập 1

□ Nhập vào số nguyên n cho tới khi nhập vào số nguyên dương thì dừng.

□ Xếp loại học lực

- Nhập điểm các môn: toán, lý và hóa
- Tính điểm trung bình và hiển thị xếp loại học lực theo yêu cầu sau:
 - Điểm < 5 : yếu
 - $5 \leq \text{điểm} < 6.5$: trung bình
 - $6.5 \leq \text{điểm} < 8$: khá
 - $8 \leq \text{điểm} < 9$: giỏi
 - Điểm ≥ 9 : xuất sắc

□Chương trình Tính thuế thu nhập

- Nhập họ và tên, lương, thưởng (theo một tháng cụ thể)
- Chương trình tính và lưu thu nhập và thuế thu nhập cùng với họ tên vào file c:\<họ tên>.txt.
- Thuế thu nhập được tính theo lũy tiến
 - Thu nhập tính thuế < 5T: 5% lương
 - Thu nhập tính thuế < 10T: 10% lương
 - Thu nhập tính thuế < 18T: 15% lương
- Mở rộng tính người phụ thuộc

ASP.NET Core Method, Array, Collection

Master II[®]
NHẤT NGHỆ

Lương Trần Hy Hiền
hyhien@gmail.com
0989.366.990



Phương thức (Method)

Phương thức

- ❑ Khai báo phương thức (hàm)
- ❑ Truyền tham số dạng **in** (ø)
- ❑ Truyền tham số dạng **out**
- ❑ Truyền tham số dạng **ref**

Khai báo phương thức

```
[modifiers] return_type MethodName([parameters])  
{  
    // Thân phương thức  
}
```

Ví dụ:

```
public static void Xuat(SinhVien sv)  
{  
    Console.WriteLine("Ma so: {0}. Ho ten: {1}", sv.MaSV, sv.HoTen);  
    //Cau lenh xuat Sinh vien  
}
```


Phương thức dạng "in"

❑ Thân phương thức **chỉ tham khảo** giá trị của tham số **không thay đổi** giá trị của tham số

❑ Ví dụ:

```
public static void Xuat(SinhVien sv)
```

```
{
```

```
    Console.Write("Ma so: {0}. Ho ten: {1}", sv.MaSV, sv.HoTen);
```

```
    //Cau lenh xuat Sinh vien
```

```
}
```

❑ Gọi hàm trong hàm Main:

```
Xuat(sv1);
```

Phương thức dạng “out”

❑ Thân phương thức cấp phát (khởi tạo) giá trị của tham số trước khi sử dụng. Ra khỏi hàm giá trị của tham số thay đổi.

❑ Ví dụ:

```
public static void Nhap(out SinhVien sv)
{
    sv = new SinhVien();
    //Cau lenh nhap sinh vien
}
```

❑ Gọi trong hàm Main:

```
Nhap(out sv2);
```

Phương thức dạng "ref"

❑ Ra khỏi hàm giá trị của tham số sẽ thay đổi

❑ Ví dụ:

```
public static void TinhDiemTrungBinh(ref StrHocSinh hs)
{
    hs.DTB = (hs.Toan + hs.Van)/2;
}
```

❑ Gọi trong hàm Main:

```
TinhDiemTrungBinh(ref hs);
```

Mảng - Array

- ❑ Mảng là tập hợp các biến có cùng kiểu dữ liệu, cùng tên nhưng có chỉ số khác nhau.
- ❑ Trong C#, mảng có chỉ số bắt đầu là 0 và luôn luôn là mảng động (mảng có khả năng thay đổi kích thước).
- ❑ Ví dụ: Mảng nguyên 5 phần tử

0	1	2	3	4	Chỉ số
10	15	20	27	2009	Giá trị

Mảng (tt)

- ❑ Các loại mảng:
 - Mảng 1 chiều
 - Mảng 2 chiều
 - Mảng nhiều chiều
 - Mảng Jagged Array

Mảng 1 chiều

□Cú pháp:

```
Kieu[] tenMang;
```

```
tenMang = new Kieu[so_phan_tu];
```

Hoặc:

```
Kieu[] tenMang = new Kieu[so_phan_tu];
```

□Ví dụ:

```
int [] mang = new int[5];
```

```
doube[] B = new double[3];
```

```
string[] C = new string[4];
```

Thao tác với mảng 1 chiều

❑ Lấy độ dài mảng:

```
int dodai = mang.Length;
```

❑ Truy cập phần tử thứ **index**: **mang[index]**

```
int kq = mang[1] + mang[2];
```

❑ Duyệt qua các phần tử trong mảng:

```
for(int i = 0; i < mang.Length; i++)
```

```
{
```

```
    //xử lý phần tử mang[i]
```

```
}
```


Bài tập 1

- ☐Viết hàm nhập mảng 1 chiều các số nguyên
- ☐Viết hàm xuất mảng 1 chiều các số nguyên
- ☐Viết hàm tính tổng các phần tử trong mảng
- ☐Viết hàm tìm số lớn nhất trong mảng 1 chiều
- ☐Viết hàm đếm số lượng số nguyên dương chẵn có trong mảng
- ☐Viết hàm main thực hiện:
 - Khai báo mảng
 - Gọi hàm nhập
 - Gọi hàm xuất
 - In kết quả tổng
 - In số lớn nhất
 - In số lượng số nguyên dương chẵn

Mảng 2 chiều

❑ Khai báo mảng 2 chiều:

```
kieu[, ] ten = new kieu[hang, cot];
```

❑ Ví dụ:

```
int[, ] Matran = new int[2,3];
```

❑ Gán giá trị:

```
Matran[0,1] = 2;
```

```
Matran[1,2] = 3;
```

Làm việc với mảng 2 chiều

□ Duyệt mảng:

```
double [, ] matrix = new double[10, 10];  
for (int i = 0; i < 10; i++)  
{  
    for (int j=0; j < 10; j++)  
        matrix[i, j] = 4;  
}
```

Mảng nhiều chiều

□ Ví dụ:

```
string[, ,] my3DArray;
```

- ❑ Mỗi mảng là một thể hiện của lớp Array.
- ❑ Các thuộc tính & phương thức của lớp Array:
 - **Length**: số phần tử (số chiều)
 - **Copy**(mang1, mang2, chieudai)
 - **Sort**(mang)
 - **Reverse**(mang)

Bài tập 2

Khai báo mảng số nguyên 2 chiều 4 dòng 5 cột.
Viết chương trình:

- ☐ Nhập vào các phần tử của mảng (giá trị = *số thứ tự dòng + số thứ tự cột*)
- ☐ In các giá trị của mảng
- ☐ In giá trị nhỏ nhất/lớn nhất trong mảng
- ☐ In tổng số các giá trị, giá trị trung bình

COLLECTION

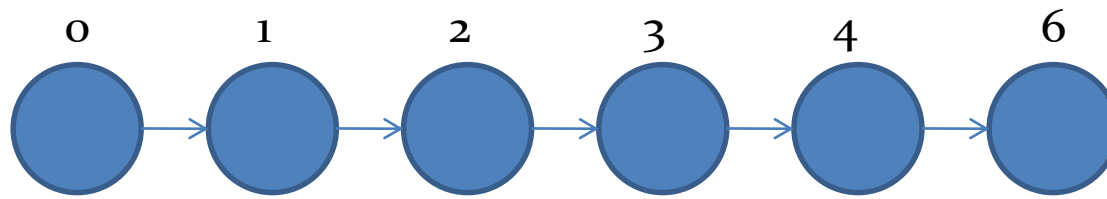
Lương Trần Hy Hiến - hyhien@gmail.com

Collections – Tập hợp

- ❑ Là cấu trúc lưu trữ các phần tử có kiểu dữ liệu khác nhau và không hạn chế số lượng phần tử.
- ❑ Các lớp có kiểu tập hợp nằm trong namespace **System.Collections** bao gồm:
 - List
 - ArrayList
 - HashTable
 - Dictionary
 - Queue
 - Stack

- MẢNG ĐỘNG
 - ArrayList
 - List
- TỪ ĐIỂN
 - Hashtable
 - Dictionary
- ĐỌC GHI ĐỐI TƯỢNG TỪ FILE
 - Đọc đối tượng từ file
 - Ghi đối tượng từ file

- ▶ Mảng động hay còn gọi là danh sách được sử dụng để quản lý danh sách các phần tử có thứ tự và hỗ trợ các thao tác thêm, xóa, sửa, tìm kiếm,...
- ▶ Hình ảnh danh sách như sau



- ▶ 2 lớp sau đây được sử dụng để tạo mảng động
 - **List<Kiểu dữ liệu>**: danh sách có kiểu
 - **ArrayList**: danh sách không kiểu

Mảng động – List

- ❑ `List<kieu_DL> mylist = new List<kieu_DL>();`
- ❑ Phải chỉ định rõ kiểu dữ liệu khi dùng. VD:
`List<string> mylist = new List<string>();`
 - ▶ Tạo mảng rỗng chứa số nguyên
 - `List<int> ages = new List<int>();`
 - ▶ Tạo mảng số thực khởi đầu 3 phần tử
 - `List<double> salaries = new List<double>()`
`{1.2, 3.4, 5.6};`

THAO TÁC MẢNG ĐỘNG - List

- Thao tác thông thường
 - **Add(<kiểu> element)**: thêm phần tử vào mảng
 - **Remove(<kiểu> element)**: xóa phần tử khỏi mảng
 - **[index]**: truy xuất
 - **Count**: lấy số phần tử trong mảng
- Tìm và kiểm tra
 - **int IndexOf(<kiểu> element)**: tìm vị trí phần tử từ đầu
 - **int LastIndexOf(<kiểu> element)**: tìm vị trí phần tử từ cuối
 - **bool Contains(<kiểu>element)**: kiểm tra sự tồn tại
- Các thao tác khác
 - **Clear()**: xóa sạch
 - **Reverse()**: đảo ngược
 - **Sort()**: sắp xếp

VÍ DỤ: MẢNG ĐỘNG - LIST

// Tạo mảng rỗng và thêm vào 3 phần tử

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

```
ages.Add(77);
```

```
ages.Add(33);
```

```
ages.Add(88);
```

Thêm các phần tử vào
mảng

```
Console.WriteLine("Số phần tử: {0}", ages.Count);
```

// Sửa phần tử thứ 2

```
ages[1] = 55;
```

// Tìm vị trí phần tử 88

```
int index = ages.IndexOf(88);
```

```
Console.WriteLine("Vị trí của 88: {0}", index);
```

// Xóa phần tử 77

```
ages.Remove(77);
```

// Duyệt và xuất tất cả các phần tử trong mảng

```
foreach (int age in ages)
```

```
{
```

```
    Console.WriteLine(" >> Phần tử: {0}", age);
```

```
}
```

// Xóa sạch các phần tử trong mảng

```
ages.Clear();
```

Mảng động – ArrayList

- ❑ Không cần chỉ định kiểu khi khai báo
- ❑ Các phần tử có thể có kiểu dữ liệu khác nhau.
 - ▶ Tạo mảng rỗng chứa phần tử kiểu bất kỳ
 - **ArrayList items = new ArrayList();**
 - ▶ Tạo mảng rỗng có khởi đầu 3 phần tử
 - **ArrayList student = new ArrayList()**
{“Ngoc Thanh”, true, 80};
 - ArrayList birds = new ArrayList();
birds.Add(“cat”);
birds.Add(10952);
foreach (Bird b in birds)
 b.ToString();

ArrayList Members

□ Some Methods:

- BinarySearch()
- IndexOf()
- Sort()
- ToArray()
- Remove()
- RemoveAt()
- Insert()

• Some Properties:

- Count
- Capacity
- IsFixedSize
- IsReadOnly
- IsSynchronized

Lớp Stack<T>

- ❑ Mảng động, kiểu tùy ý, kích thước tự động, thêm/bớt xảy ra tại đỉnh (LIFO)
- ❑ Phương thức:
 - **Push(T)** – thêm phần tử vào đỉnh stack
 - **Pop()** – xóa phần tử ở đỉnh stack và trả về giá trị đó

Stack<T> – Ví dụ

❑ Sử dụng **Push()**, **Pop()** và **Peek()**

```
static void Main()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("1. Ivan");
    stack.Push("2. Nikolay");
    stack.Push("3. Maria");
    stack.Push("4. George");
    Console.WriteLine("Top = {0}", stack.Peek());
    while (stack.Count > 0)
    {
        string personName = stack.Pop();
        Console.WriteLine(personName);
    }
}
```

Lớp Queue<T>

- ❑ Mảng động, kiểu tùy ý, kích thước động; thêm/bớt xảy ra 2 chiều (FIFO)
- ❑ Phương thức:
 - **Enqueue(T)** – thêm phần tử vào cuối mảng
 - **Dequeue()** – lấy phần tử đầu mảng và trả về giá trị đó

Ví dụ Queue

```
static void Main()
{
    Queue<string> queue = new
Queue<string>();

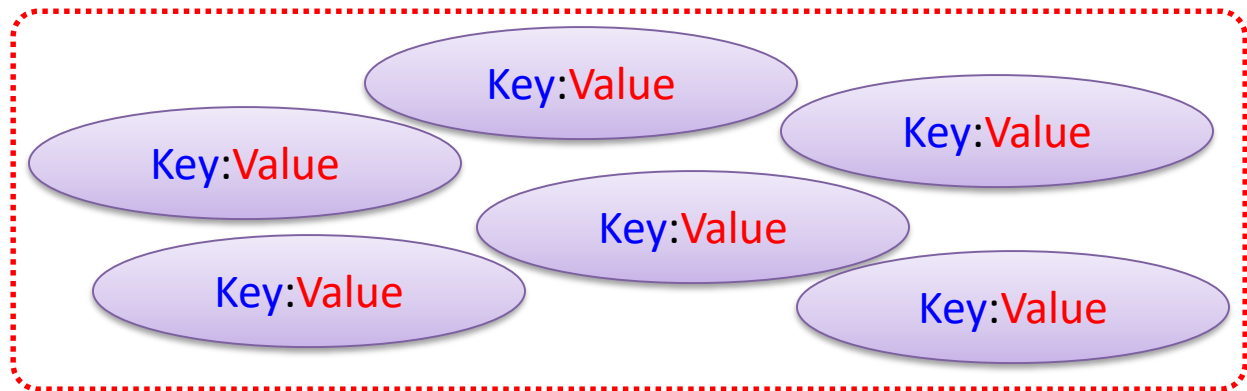
    queue.Enqueue("Message One");
    queue.Enqueue("Message Two");
    queue.Enqueue("Message Three");
    queue.Enqueue("Message Four");
    while (queue.Count > 0)
    {
        string message = queue.Dequeue();
        Console.WriteLine(message);
    }
}
```

THAO TÁC MẢNG KHÔNG KIỂU

- ❑ Các hoạt động thao tác mảng không kiểu hoàn toàn tương tự mảng có kiểu chỉ khác duy nhất kiểu của các phần tử thao tác là **object** thay vì **<kiểu>** được chỉ định bởi người dùng.
- ❑ Ví dụ:
 - Add(**<kiểu>** element) đối với mảng có kiểu
 - Add(**object** element) đối với mảng không kiểu
- ❑ Lưu ý:
 - Khi truy xuất phần tử, bạn cần ép trở lại kiểu của nó.
Ví dụ:
 - **String** HoTen= (**String**)MyArrayList[5];

- Dùng quản lý tập hợp các phần tử không phân biệt thứ tự. Mỗi phần tử gồm 2 phần (khóa và giá trị). Để truy xuất giá trị của một phần tử ta phải biết khóa của nó.

- Hình ảnh



- 2 lớp thường dùng

- **Dictionary**<Kiểu khóa, Kiểu giá trị>: có kiểu
- **Hashtable**: không kiểu

KHỞI TẠO TỪ ĐIỂN

```
// Tạo từ điển không kiểu, thêm vào 3 phần tử
Hashtable student = new Hashtable();
student.Add("Name", "Nguyen Van Teo");
student.Add("Age", 30);
student.Add("Gender", true);
// Tạo từ điển có kiểu, thêm 4 phần tử
Dictionary<String, Double> emp = new Dictionary<String, Double>();
emp.Add("Nguyen Thanh Tin", 8.5);
emp.Add("Pham Thi Hoa", 7);
emp.Add("Ngo Quoc Bao", 6.5);
emp.Add("Luong Van Thanh", 9);
// Tạo từ điển có kiểu, khởi đầu 3 phần tử
Dictionary<String, String> words = new Dictionary<String, String>()
{
    {"Love", "Yêu"},
    {"One", "Một"},
    {"School", "Trường học"}
};
```

THAO TÁC TỪ ĐIỂN

- Truy xuất
 - **[Key]**: truy xuất giá trị của phần từ
- Thuộc tính thường dùng
 - **Count**: lấy số phần tử
 - **Keys**: lấy tập hợp khóa
 - **Values**: lấy tập hợp giá trị
- Phương thức thường dùng
 - **Add(Key, Value)**: thêm một phần từ
 - **Remove(Key)**: xóa một phần tử
 - **Clear()**: xóa sách
 - **ContainsKey(Key)**: kiểm tra sự tồn tại của khóa
 - **ContainsValue(Value)**: kiểm tra sự tồn tại của giá trị

VÍ DỤ TỪ ĐIỂN

```
// Tạo và khởi đầu từ điển
Dictionary<String, Double[]> marks = new Dictionary<String, Double[]>()
{
    {"Tuấn", new Double[]{7.0, 8.5, 9.5}},
    {"Hoa", new Double[]{5, 7, 4}},
    {"Hồng", new Double[]{6, 8, 10}}
};
// Sửa phần tử thứ hai trong mảng có khóa là "Hoa"
marks["Hoa"][2] = 5.5;
//Duyệt và xuất ra thông tin của mỗi phần tử
foreach (String Name in marks.Keys)
{
    Double[] MA = marks[Name];
    Console.WriteLine("{0}, {1}, {2}, {3}", Name, MA[0], MA[1], MA[2]);
}
Console.WriteLine("Số phần tử: {0}", marks.Count);
```


HashTable

- ❑ Là Kiểu từ điển
- ❑ Mỗi phần tử bao gồm 01 cặp [key-value]
- ❑ Truy xuất nhanh.
- ❑ Các cặp key không trùng nhau

```

Hashtable ht = new Hashtable();
ht.Add("masp", "SP001");
ht.Add("soluong", 10);
ht.Add("gia", 123.45);
foreach (DictionaryEntry de in ht)
{
    s += string.Format("kqy={0}, value = {1}",
        de.Key, de.Value);
}
    
```

Key	Value
masp	SP001
soluong	10
gia	123.45

Bài tập 1

- ☐ Nhập danh sách hàng hóa (mã, tên, giá)
- ☐ Tìm kiếm hàng hóa theo mã
- ☐ Chỉnh sửa hàng hóa tìm được
- ☐ Tìm kiếm hàng hóa theo tên (chứa)
- ☐ Tìm kiếm hàng hóa theo giá (min, max)
- ☐ Hướng dẫn:
 - Dùng 3 danh sách để lưu trữ dữ liệu (List<int>, List<string>, List<double>)

Bài tập 2

❑ Thực hiện lại bài ở demo1 theo cách sau

- Sử dụng ArrayList để lưu thông tin hàng hóa (mã, tên, giá)
- Sử dụng List<ArrayList> để lưu danh sách hàng hóa

Bài tập 3

☐ Xây dựng một từ điển đơn giản

- Khởi đầu 10 từ
- Cho phép tra cứu từ
- Cho phép nhập thêm từ vào

☐ Thực hiện lại bài demo 2 theo hướng dẫn

- Dùng Hashtable thay cho ArrayList để lưu thông tin hàng hóa (mã, tên, giá)
- Sử dụng Dictionary< Hashtable> để quản lý hàng hóa

