

Chương 1: GIỚI THIỆU CÔNG NGHỆ .NET VÀ NGÔN NGỮ C#

Giảng viên: Vũ Minh Sang
Phòng: E9.4
E-mail: sangvm@uit.edu.vn

Nội dung



Công nghệ .NET




Ngôn ngữ C#



Công nghệ .NET

- 📖 Giới thiệu Công nghệ .NET
- 📖 Các framework trong .NET

Giới thiệu Công nghệ .NET

 Công nghệ .NET (dot Net) là tên gọi chung các công nghệ phát triển ứng dụng của Microsoft

 Điểm nổi bật của .NET




- Tính bảo mật cao
- Tăng hiệu suất
- Giảm chi phí phát triển phần mềm
- Sử dụng được cho đa nền tảng.
- Dễ dàng tích hợp với hệ thống cũ

 Công nghệ .NET có các nền tảng sau:

- .NET Framework
- .NET Core
- Xamarin

Giới thiệu Công nghệ .NET

Phân biệt platform, framework, library

-  Platform: liên quan đến phần cứng và phần mềm
-  Framework: thường xây dựng bên trên platform và tích hợp trong platform; đồng thời là cái khung để xây dựng và phát triển phần mềm.
-  Library: cung cấp các hỗ trợ về chức năng cho phát triển phần mềm. Thông thường nằm trong framework hoặc platform

Giới thiệu Công nghệ .NET

“.NET Framework là một nền tảng (platform) cho phát triển ứng dụng trên windows”

Như vậy: .NET Framework vừa là một platform, vừa là một framework và cũng là một library.

Lý do: .NET Framework

- Tạo ra khung cho phát triển ứng dụng (framework)
- Cung cấp hệ thống thư viện (library)
- Hoạt động trong môi trường máy ảo CLR

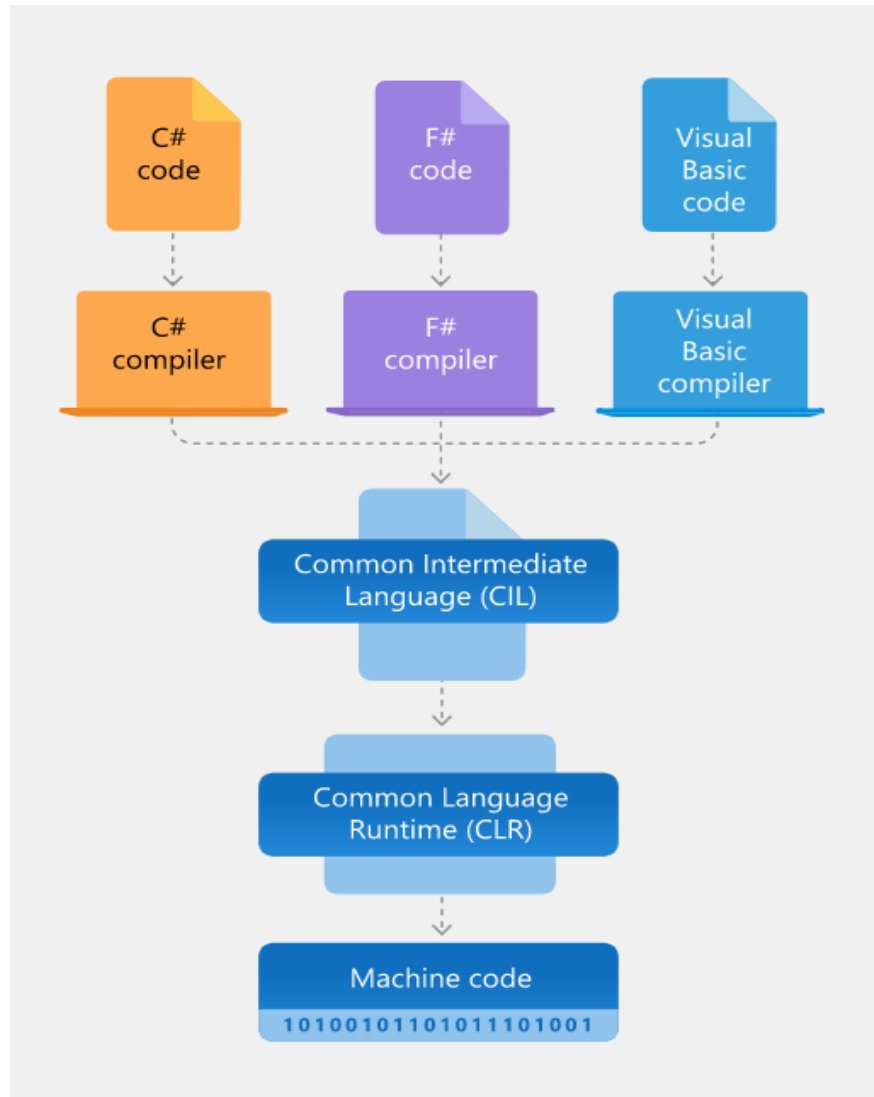
.NET Framework

- ❏ Là một nền tảng dành cho việc phát triển ứng dụng desktop hoặc web chạy trên windows.
- ❏ Phiên bản đầu tiên .NET Framework 1.0 (2002). Phiên bản mới nhất .NET Framework 4.8 (2019)
- ❏ Tạo ra khung sườn cho nhiều loại ứng dụng khác nhau:
 - Ứng dụng desktop Windows Forms, Windows Presentation Foundation (WPF)
 - Ứng dụng web ASP.NET
 - Ứng dụng hướng dịch vụ Windows Communication Foundation (WCF)
- ❏ Cung cấp các bộ thư viện cho việc xây dựng ứng dụng
- ❏ Các thư viện để kết nối CSDL: ADO.NET, Entity Framework

Đặc điểm .NET Framework

- 📖 **Khả năng tương tác:** cung cấp nhiều hỗ trợ cho các phiên bản cũ hơn.
- 📖 **Linh động:** ứng dụng được xây dựng trên .NET Framework có thể làm việc trên bất cứ nền tảng nào của Windows.
- 📖 **Bảo mật:** Cơ chế bảo mật tốt, giúp xác nhận và xác minh các ứng dụng. Ứng dụng có thể xác định rõ ràng cơ chế bảo mật cho riêng mình. Cơ chế bảo mật có thể cấp quyền cho người dùng truy cập vào mã hoặc chương trình đang chạy.
- 📖 **Quản lý bộ nhớ:** CLR thực hiện tất cả công việc hoặc quản lý bộ nhớ của ứng dụng. Có **Garbage Collector** được dùng để giải phóng tài nguyên của ứng dụng khi không sử dụng.
- 📖 **Triển khai đơn giản:** có các công cụ để đóng gói các ứng dụng xây dựng trên .NET Framework. Sau đó, các gói này có thể được phân phối đến người dùng và sẽ tự động được cài đặt vào máy người dùng.

.NET Framework



Kiến trúc .NET Framework

Nguồn: (<https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>)

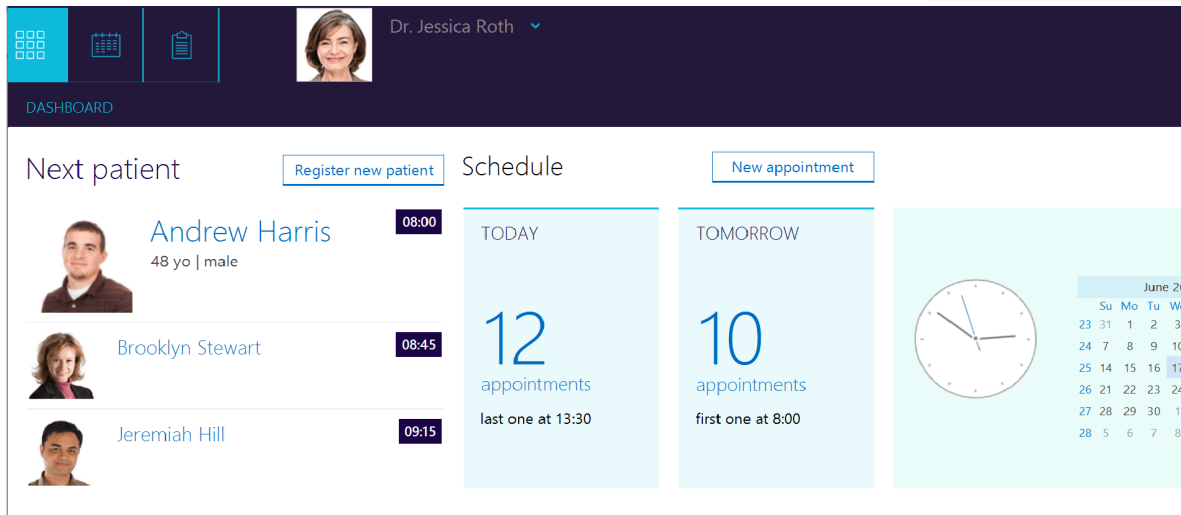
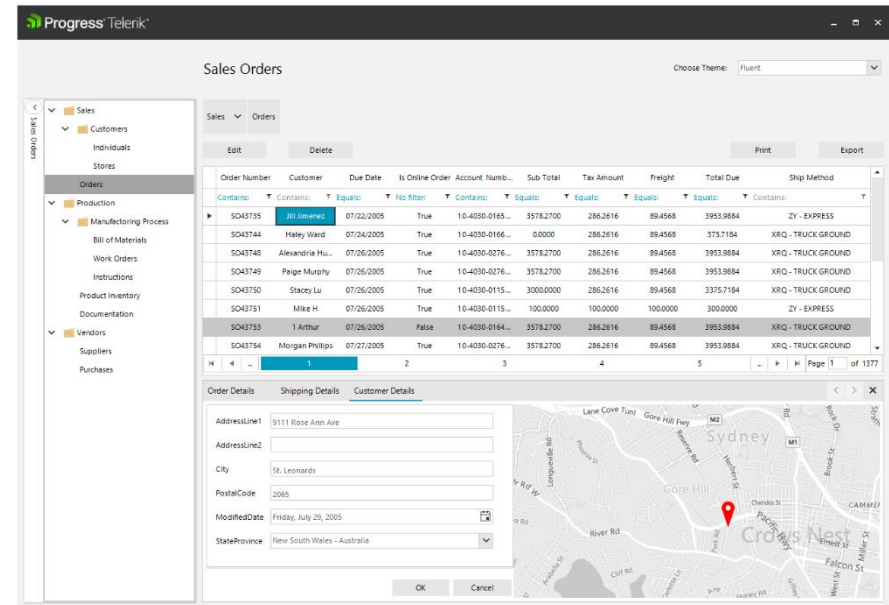
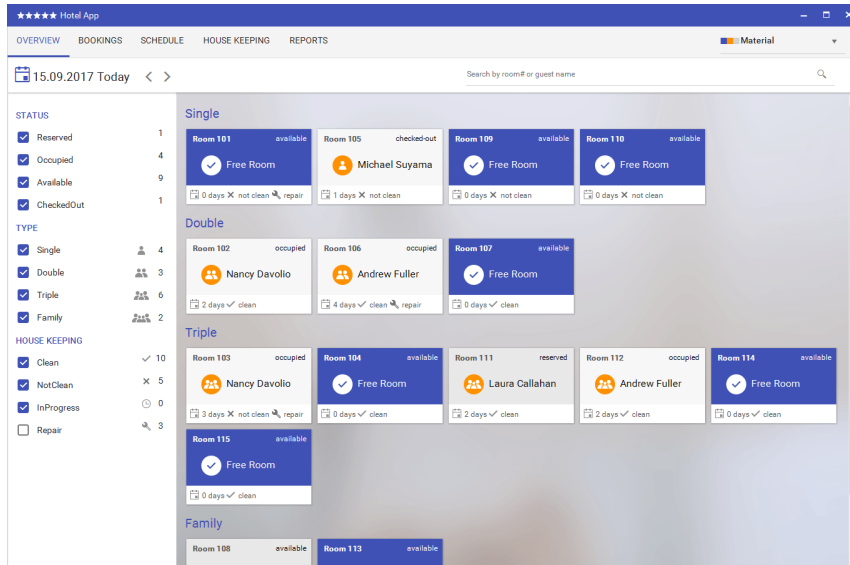
Các framework trong .NET Framework

Windows Forms (Winforms):

- 📖 Là framework cho phát triển ứng dụng desktop cho Windows, ra đời từ .NET Framework 1.0.
- 📖 Đơn giản hóa lập trình GUI (giao diện đồ họa người dùng), thiết kế trực quan đơn giản, không cần phải tự viết code nhiều, phù hợp cho người mới đầu học hoặc tự học.
- 📖 Mô hình lập trình đơn giản, dễ nắm bắt, nhận được sự hỗ trợ tốt từ cộng đồng và các hãng thứ 3 (Devexpress, Syncfusion, Telerik...
- 📖 Tuy nhiên, không phù hợp với yêu cầu hiện nay.
- 📖 Vì vậy, có thể sử dụng như một công cụ lập trình nhanh cho các ứng dụng nhỏ hoặc xây dựng các prototype.

Các framework trong .NET Framework

Windows Forms (Winforms)



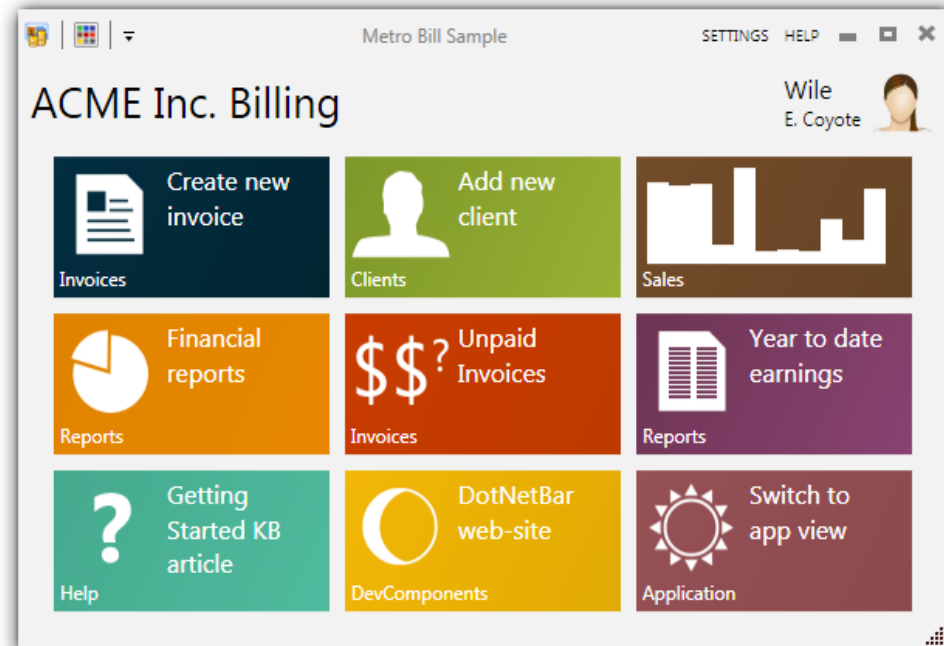
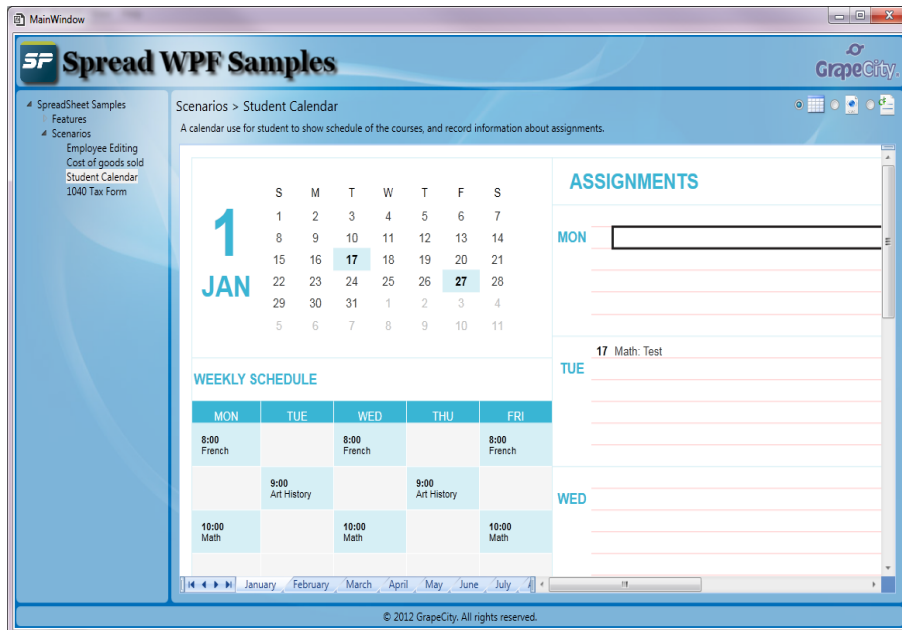
Các framework trong .NET Framework

Windows Presentation Foundation (WPF):

- ❏ Ra đời ở .NET Framework 3.5.
- ❏ Sử dụng DirectX để tạo giao diện với khả năng xử lý đồ họa mạnh, cho phép tạo ra giao diện hiện đại, đẹp mắt, mượt mà đi kèm với các hiệu ứng mà winforms khó có thể làm được.
- ❏ Sử dụng ngôn ngữ XAML để biểu diễn giao diện.
- ❏ Độc lập việc thiết kế giao diện với code xử lý chức năng của giao diện
- ❏ Sử dụng kiến trúc MVVM (Model-View-ViewModel)
- ❏ Cơ chế Data Binding mạnh mẽ, giúp hiển thị dữ liệu hiệu quả, ổn định và an toàn.
- ❏ Nhược điểm: phức tạp, khó học, khó sử dụng thành thạo.
- ❏ Phù hợp với các dự án lớn đòi hỏi giao diện đẹp, hiện đại.

Các framework trong .NET Framework

Windows Presentation Foundation (WPF):



Các framework trong .NET Framework

ASP.NET:

- 📖 Là framework được sử dụng để phát triển các ứng dụng dựa trên nền web
- 📖 Ứng dụng web sẽ được xử lý trên một máy chủ web IIS (Internet Information Services) của Windows.
- 📖 Có các mô hình lập trình khác nhau trên nền ASP.NET: ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web API và một số mô hình khác như: SignalR, Web Pages, WebHooks

Các framework trong .NET Framework

ASP.NET Web Forms:

- ❏ Là mô hình lập trình ứng dụng web đầu tiên của ASP.NET.
- ❏ Xây dựng mô hình phát triển “stateful” (gần giống ứng dụng trên desktop thay cho bản chất stateless của web và HTTP).
- ❏ Vì vậy, mô hình này rất nặng nề, mã HTML tự sinh ra không kiểm soát được và tách xa tiêu chuẩn về web.

Các framework trong .NET Framework

ASP.NET MVC:

- 📖 Là mô hình phát triển ứng dụng web về với các tiêu chuẩn chung của web.
- 📖 Phát triển ứng dụng web theo mô hình MVC.
- 📖 Nhiều thư viện hỗ trợ.

Các framework trong .NET Framework

ASP.NET Web API:

- 📖 Là framework hỗ trợ xây dựng thành phần dịch vụ (services) cho hệ thống phần mềm hướng dịch vụ (Service-Oriented Application – SOA).
- 📖 Thường sử dụng song song với MVC để bổ sung API cho hệ thống.

Các framework trong .NET Framework

ASP.NET Web Pages:

- 📖 Là framework đơn giản gọn nhẹ giúp xây dựng các trang web động.
- 📖 Cách thức hoạt động gần với các trang web động viết bằng PHP hoặc ASP.
- 📖 Sử dụng code C# kết hợp với HTML
- 📖 Mô hình lập trình đơn giản và phù hợp với các project cỡ nhỏ.

Các framework trong .NET Framework

SignalR:

- 📖 Là thư viện giúp server gửi thông báo bất đồng bộ theo thời gian thực tới web client.
- 📖 Ví dụ: nếu có dữ liệu mới xuất hiện ở server thì server có thể gửi trực tiếp dữ liệu này cho client mà không cần chờ client gửi yêu cầu.
- 📖 Giúp xây dựng các ứng dụng web có tính tương tác cao theo thời gian thực.
- 📖 Thay đổi hoàn toàn mô hình request/response truyền thống.
- 📖 Gồm 2 thành phần: server chạy trên ASP.NET và Javascript trên client

Các framework trong .NET Framework

ADO.NET:

- 📖 Giúp ứng dụng (desktop hoặc web) kết nối và làm việc với các hệ quản trị CSDL quan hệ (SQL Server, MySQL...) hoặc các nguồn dữ liệu khác: OLE DB, ODBC...
- 📖 SQL Server được ADO.NET hỗ trợ trực tiếp, các hệ quản trị khác (MySQL) được hỗ trợ thông qua connector của bên thứ ba.

Các framework trong .NET Framework

Entity Framework:

- ☐ Là một ORM (Object – Relational Mapping) cho phép ánh xạ giữa các object (của một class) với các bản ghi của một bảng CSDL.
- ☐ Lập trình CSDL là cách xử lý các đối tượng của ngôn ngữ lập trình hướng đối tượng. Cho nên không cần quan tâm đến ngôn ngữ SQL.
- ☐ Toàn bộ thao tác liên quan đến dữ liệu đều do Entity Framework đảm nhiệm.
- ☐ Được sử dụng rộng rãi cho cả ứng dụng desktop và web

Ngôn ngữ lập trình trong .NET Framework

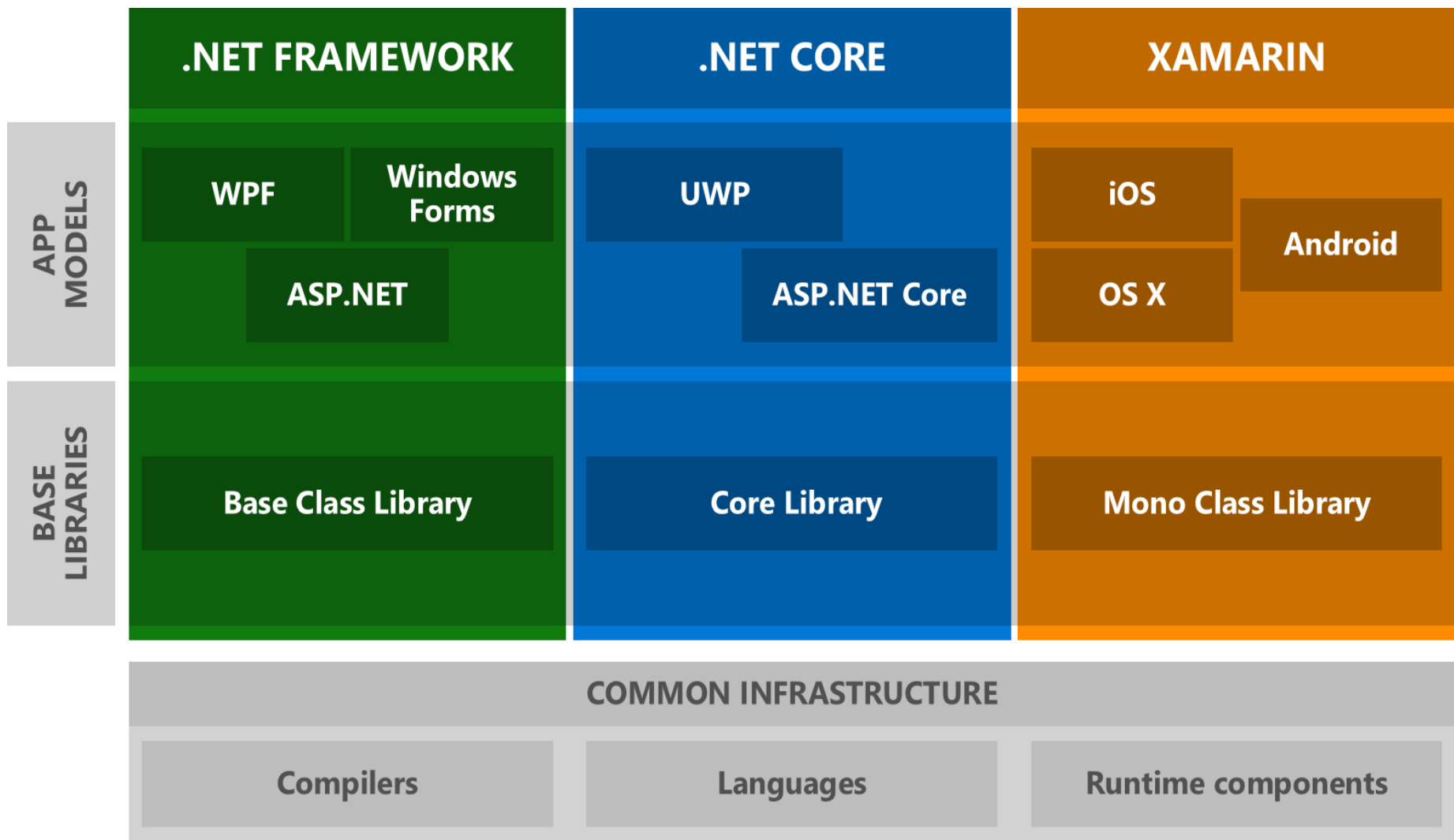
- 📖 **Visual C#:** được xây dựng riêng cho .NET Framework, kế thừa cấu trúc cú pháp và nhiều đặc điểm của C/C++.
- 📖 **Visual C++ for CLI:** đây là ngôn ngữ C++ dịch sang mã CIL (mã trung gian của .NET Framework) thay thế cho mã native như Visual C++ cũ.
- 📖 **Visual Basic .NET:** là ngôn ngữ kế thừa của ngôn ngữ Visual Basic.
- 📖 **Visual F#:** là ngôn ngữ lập trình hàm (Functional Programming). Phù hợp để xây dựng thư viện hàm cho xử lý thông tin.

.NET Core

- 📖 Đây là công nghệ mới của Microsoft được giới thiệu vào năm 2016 với phiên bản Core 1.0 đi kèm với Visual Studio 2015; phiên bản mới nhất là Core 3.11 (2020).
- 📖 Cùng nguyên lý và ý tưởng với .NET Framework nhưng hoạt động trên đa nền tảng, tính module hóa và hiệu suất cao hơn.
- 📖 Ứng dụng trên .NET Core có thể triển khai trên Linux, Mac OS hoặc Windows.
- 📖 .NET Core cũng cung cấp khung sườn cho phát triển ứng dụng trên desktop: Windows Forms, từ .NET Core 3.1 có thêm WPF; cho ứng dụng phát triển web ASP.NET Core (MVC, API, Razor Pages, Blazor)
- 📖 Cung cấp hệ thống thư viện cho việc phát triển ứng dụng.
- 📖 Ngôn ngữ cho .NET Core: Visual C#, Visual Basic.NET và Visual F#.

Xamarin

- 📖 Là một dạng nền tảng trong công nghệ .NET, được dùng để triển khai ứng dụng trên các hệ điều hành di động: Android và iOS.
- 📖 Cho phép viết code một lần nhưng biên dịch ứng dụng sang cả hai nền tảng di động.



Các nền tảng trong công nghệ .NET



Ngôn ngữ C#



Ngôn ngữ C#

- 📖 Giới thiệu ngôn ngữ C#
- 📖 Cú pháp cơ bản
- 📖 Biến và hằng
- 📖 Các kiểu dữ liệu
- 📖 Toán tử (operator)
- 📖 Cấu trúc điều khiển
- 📖 Mảng
- 📖 Kiểu liệt kê (Enumeration)
- 📖 Kiểu cấu trúc (Struct).
- 📖 Namespace



Giới thiệu ngôn ngữ C#

Giới thiệu ngôn ngữ C#

- ❏ Là một ngôn ngữ lập trình được xây dựng riêng cho .NET Framework.
- ❏ Phiên bản đầu tiên 1.0 ra đời với .NET Framework 1.0 vào năm 2002; hiện nay là phiên bản 8.0.
- ❏ Được thiết kế dựa trên các ngôn ngữ lập trình C/C++, Java.
- ❏ Có cú pháp, các cấu trúc điều khiển, một số kiểu dữ liệu cơ sở... giống với C/C++, Java
- ❏ Là ngôn ngữ lập trình hoàn toàn hướng đối tượng, mọi thứ trong C# đều là class.
- ❏ Không có khái niệm hàm toàn cục, biến toàn cục.
- ❏ Một số IDE hỗ trợ các công nghệ .NET: Visual Studio, Visual Studio Code, MonoDevelop, Morfik (cho phát triển ứng dụng web), Turbo C#....




Cú pháp cơ bản

Cú pháp cơ bản

Câu lệnh (statement):

- Phân biệt chữ hoa, chữ thường
- Không sử dụng các khoảng trắng trong các khai báo.
- Sử dụng dấu chấm phẩy ";" để kết thúc câu lệnh

 **VD:** `Console.WriteLine("Hello world");`
`Console.ReadKey();`

Một số loại câu lệnh trong C#:

- *Lệnh khai báo (Declaration statements):* dùng để khai báo biến và hằng.
- *Lệnh tính toán (Expression statements):* thường gọi là biểu thức, dùng để thực hiện tính toán trên dữ liệu và có thể gán giá trị về cho biến
- *Lệnh lựa chọn (Selection statements):* dùng trong các cấu trúc rẽ nhánh: if, else, switch, case
- *Lệnh lặp (Iteration statements):* thực hiện các câu lệnh lặp: do, for, foreach, in, while.

Cú pháp cơ bản

Khối lệnh:

- Một chuỗi câu lệnh tạo thành nhóm với với nhau gọi là khối lệnh (code block hoặc statement block)
- Được đặt trong cặp dấu ngoặc {}
- Các khối lệnh có thể lồng nhau

○ VD:

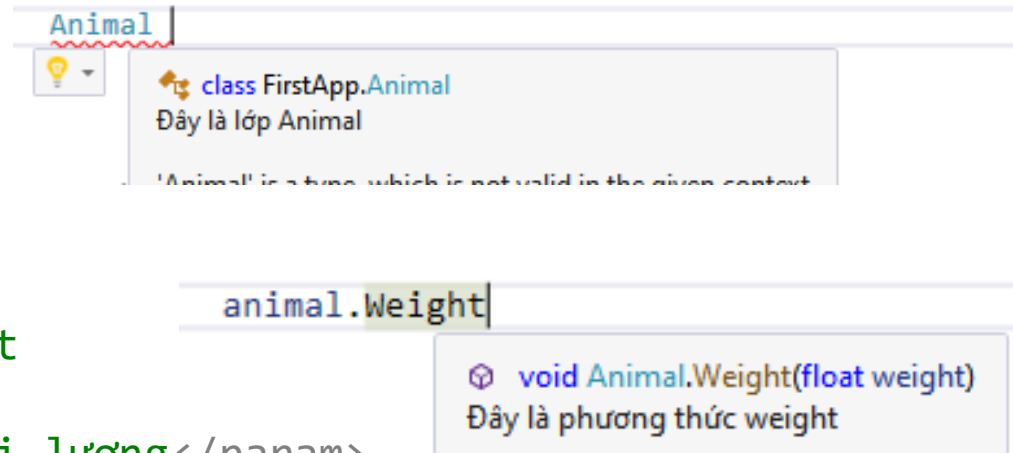
```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello world");
        Console.ReadKey();
    }
}
```


Cú pháp cơ bản

Ghi chú:

- Ghi chú trên một dòng: dùng cặp dấu "//": //Ghi chú một dòng
- Ghi chú trên nhiều dòng: dùng cặp dấu /* và */
/*
* Đây là ghi chú nhiều dòng
* Ghi rõ ý nghĩa của câu lệnh
*/
- Ghi chú tài liệu (documentation comment): cho phép tạo ra một dạng hướng dẫn sử dụng của class, phương thức.... Được Visual Studio tự sinh ra khi gõ "///" trước đối tượng cần ghi chú và sẽ hiển thị thông qua Intellisense.

```
/// <summary>  
/// Đây là lớp Animal  
/// </summary>  
public class Animal{  
    /// <summary>  
    /// Đây là phương thức weight  
    /// </summary>  
    /// <param name="weight">Khối lượng</param>  
    public void Weight(float weight){}  
}
```



Cú pháp cơ bản

📖 Từ khóa (Keyword): là những từ được C# gán cho ý nghĩa xác định, là cú pháp của ngôn ngữ. Không được dùng từ khóa để làm tên (định danh) cho biến hoặc phương thức, class....

📖 Danh sách các từ khóa trong C#:

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

Cú pháp cơ bản

📖 Định danh: là chuỗi ký tự được dùng để đặt tên cho các thành phần: biến, hằng, lớp, phương thức, tham số....

📖 Quy tắc đặt định danh:

- Chữ các chữ cái a-z, A-Z, chữ số 0-9, dấu gạch chân “_” và ký tự @.
- Không cho phép bắt đầu bằng chữ số.
- Ký tự @ chỉ cho phép đứng đầu định danh.
- Phân biệt ký tự hoa và thường.

📖 Quy ước đặt định danh: 2 loại thường dùng:

PascalCase:

- Ký tự đầu tiên bắt đầu bằng chữ hoa, nếu có nhiều từ thì viết hoa chữ cái đầu tiên của mỗi từ.
- Thường để đặt tên class, struct, enum, tên phương thức, tên các biến, đặc tính kiểu public.
- VD: `class SinhVien, public string HoTen;`

Cú pháp cơ bản

camelCase:

- Ký tự đầu tiên là chữ thường. Chữ cái đầu của các từ tiếp theo viết hoa
- Thường để đặt tên biến cục bộ, tham số của phương thức, biến thành viên của struct/class.
- VD: `public int count;` `class SinhVien(string hoTen)`

Lớp Program và phương thức Main()

- Trong C# bắt buộc phải có ít nhất một lớp.
- Program** là lớp được C# tự động sinh khi tạo project.
- Có thể đổi thành bất kỳ tên nào cũng được (theo quy tắc đặt định danh)
- Trong Program có một phương thức đặc biệt: **Main()**

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello world!");
        Console.ReadKey();
    }
}
```

- Main(): là phương thức đầu tiên được gọi khi chạy chương trình C#, là điểm khởi đầu của chương trình (còn gọi là entry point).
- Bắt buộc bắt đầu bằng `static void Main()` hoặc `static int Main()`. Phần tham số không bắt buộc



Biến và hằng

Biến

📖 Được dùng để lưu trữ thông tin, giá trị của chương trình và có thể thay đổi giá trị được.

📖 Có 3 loại biến trong C#:

- *Biến cục bộ (local variable)* : là biến nằm trong phương thức.
- *Biến thành viên*: là biến nằm trong class hoặc struct
- *Tham số*: biến để truyền giá trị vào trong phương thức.

📖 Biến được khai báo với cú pháp:

<kiểu dữ liệu> <tên biến>;

- *Kiểu dữ liệu (datatype)*: tên loại dữ liệu mà biến có thể lưu trữ.
- *Tên biến (identifier)*: là định danh của biến, được đặt theo quy tắc đặt định danh và quy ước camelCase.
- VD: **int** count;
- Bắt buộc phải gán giá trị trước khi sử dụng: **count = 0;**
- Có thể vừa khai báo biến vừa gán giá trị: **int** count = 0;
- Có thể khai báo và gán giá trị cho nhiều biến cùng kiểu:
int count = 0 , sum = 10;

Khởi tạo biến

- ❏ C# bắt buộc mọi biến phải được khởi tạo giá trị trước khi sử dụng.
- ❏ Biến thành viên nếu không được gán giá trị trực tiếp thì trình biên dịch C# sẽ tự động gán giá trị mặc định cho từng kiểu dữ liệu (giá trị 0 cho kiểu *int*).
- ❏ Biến cục bộ bắt buộc phải gán giá trị trước khi sử dụng.

```
int count;  
Console.WriteLine("{0}", count);  
Console.ReadKey();
```

[?] (local variable) `int count`
Use of unassigned local variable 'count'

Từ khóa var

📖 Khai báo biến với từ khóa **var**, biến nhận giá trị khởi tạo ban đầu theo kiểu nào thì trình biên dịch sẽ xác định biến mang kiểu dữ liệu đó.

```
var count = 0;
```

- 📖 Khi dòng lệnh trên được biên dịch, sẽ hiểu là: **int** count = 0;
- 📖 Biến phải được khởi tạo ngay lúc khai báo.

```
var count;  
count = 0;
```

🔗 (local variable) var count

Implicitly-typed variables must be initialized

Phạm vi biến

- 📖 Phạm vi (scope): là vùng câu lệnh (khối code – code block) mà biến có thể truy xuất. Được xác định như sau:
 - Biến cục bộ phạm vi tác dụng là khối code mà nó được khai báo. Khối code có thể lồng nhau, biến khai báo của khối code lớn (nằm ngoài) sẽ có phạm vi bao trùm cả khối code con bên trong.
 - Trong phạm vi của một biến, không thể khai báo biến khác trùng tên.

```
static void Main(string[] args)
{
    int count = 0;
    Console.WriteLine(count);
    {
        int sum = 10;
        Console.WriteLine(count);
        Console.WriteLine(sum);
    }

    Console.WriteLine(count);
    Console.WriteLine(sum);
}
```

The name 'sum' does not exist in the current context

Show potential fixes (Alt+Enter or Ctrl+.)

Hằng

- ❏ Hằng cũng được dùng để lưu trữ thông tin, giá trị như biến, nhưng không thể thay đổi giá trị được.
- ❏ Việc đặt tên và sử dụng tương tự như biến.
- ❏ Bắt buộc phải gán giá trị lúc khai báo.
- ❏ Cú pháp khai báo:

```
const <kiểu dữ liệu> <tên biến> = giá trị;  
const int MAX_VALUE = 100;
```

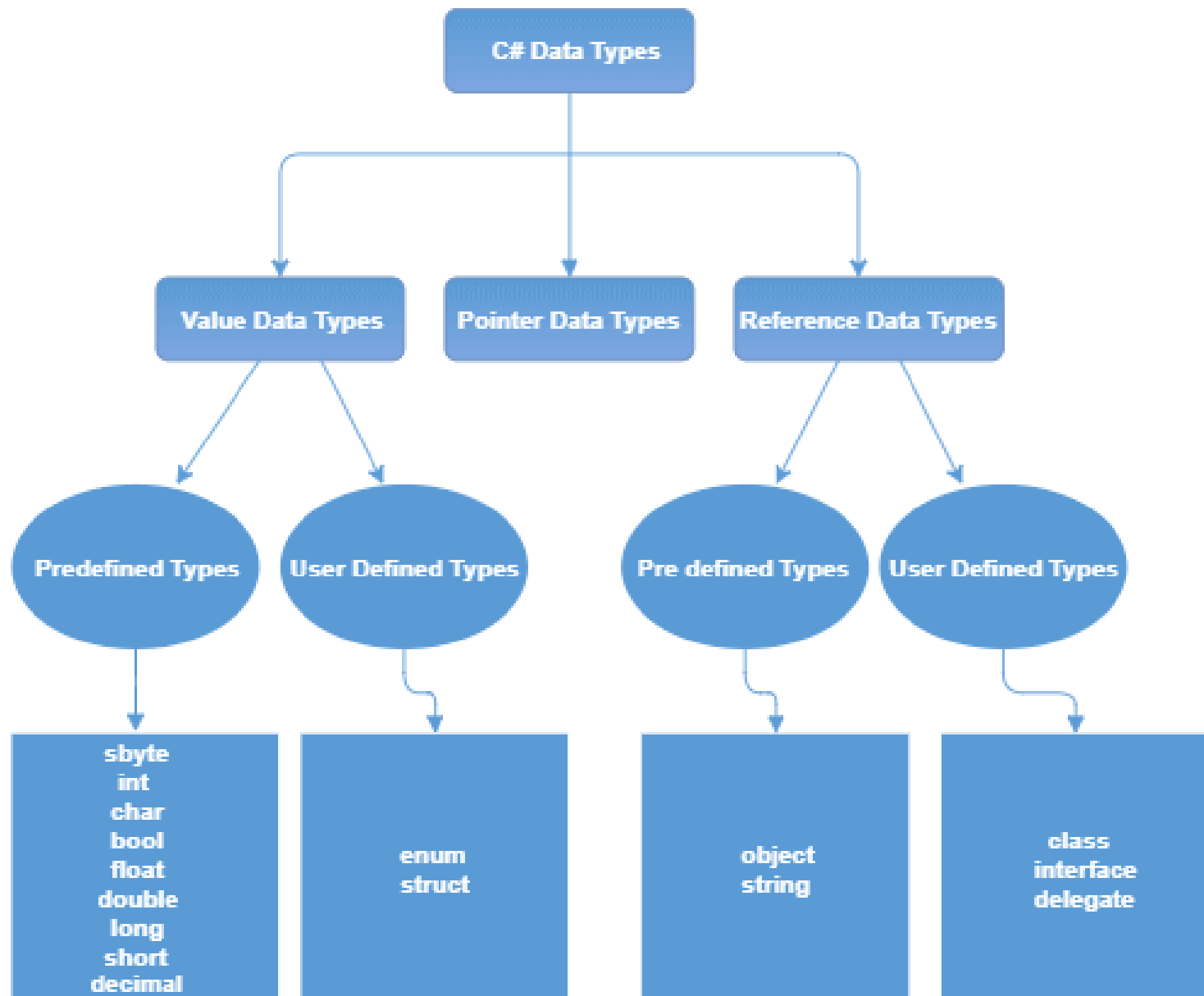


Kiểu dữ liệu

Kiểu dữ liệu

- 📖 Kiểu dữ liệu (data type) trong C# có 2 loại: *kiểu dữ liệu dựng sẵn (build-in)* do ngôn ngữ C# cung cấp và *kiểu dữ liệu do người dùng định nghĩa (user – defined)*.
- 📖 Mỗi kiểu trên phân thành 2 loại: *kiểu dữ liệu giá trị (value type)* và *kiểu dữ liệu tham chiếu (reference type)*.
- 📖 Kiểu dữ liệu giá trị kế thừa từ class **System.<ValueType>**. VD: kiểu **int** thuộc **System.Int32**.
- 📖 C# cung cấp các kiểu dữ liệu cơ sở:
 - Kiểu số nguyên.
 - Kiểu số thực
 - Kiểu logic
 - Kiểu ký tự
 - Kiểu string
 - Kiểu object

Kiểu dữ liệu



Phân biệt bộ nhớ Stack và Heap

📖 **Đều là các vùng nhớ trong RAM.**

📖 **Stack:**

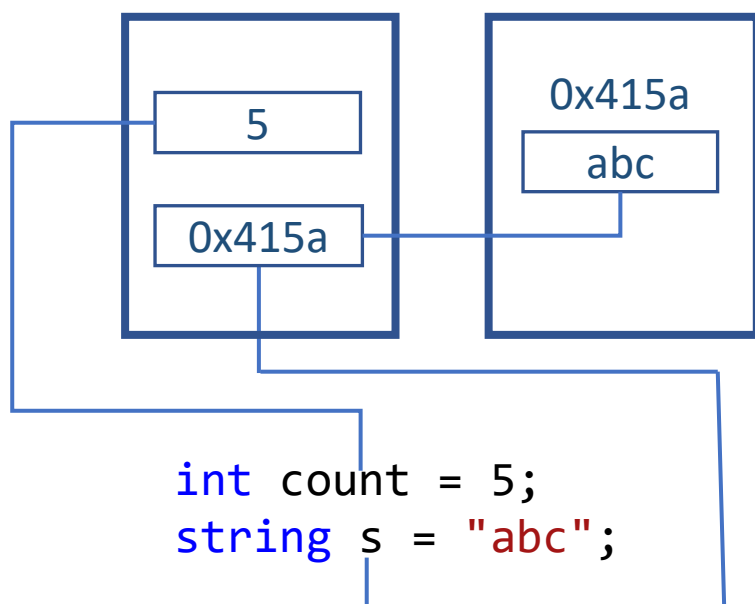
- Hoạt động theo mô hình LIFO (vào sau, ra trước).
- Lưu trữ các biến cục bộ.
- Khi biến được khai báo, giá trị của biến được lưu trữ trong stack (push)
- Khi phương thức kết thúc, tất cả biến do phương thức đó tạo ra đều bị giải phóng (pop).
- Tốc độ đọc ghi rất cao nhưng kích thước bị giới hạn.

📖 **Heap:**

- Là vùng nhớ khác stack; cho phép tự do lưu trữ giá trị.
- Giá trị được lưu trong heap không giới hạn về kích thước mà chỉ phụ thuộc và kích thước Ram.
- Tốc độ đọc ghi chậm hơn stack.
- Chương trình C# và .NET sẽ xin cấp phát và giải phóng vùng nhớ trên heap thông qua Garbage Collector.

Kiểu giá trị và kiểu tham chiếu

- ❏ Kiểu dữ liệu dựng sẵn trong C# và .NET phân loại thành 2 nhóm dựa trên việc cấp và quản lý bộ nhớ cho biến: kiểu giá trị và kiểu tham chiếu.
- ❏ Kiểu giá trị: lưu dữ liệu trực tiếp trong biến và lưu trong stack.
- ❏ Kiểu tham chiếu: dữ liệu được lưu trong heap, biến là vùng nhớ trong stack và chứa địa chỉ của vùng nhớ của heap.



- ❏ Với các kiểu dữ liệu cơ sở:
 - Kiểu *object* và *string* thuộc kiểu tham chiếu.
 - Các kiểu còn lại thuộc kiểu giá trị

Kiểu số nguyên

NAME	.NET TYPE	DESCRIPTION	RANGE (MIN:MAX)
sbyte	System.SByte	8-bit signed integer	-128:127 ($-2^7:2^7-1$)
short	System.Int16	16-bit signed integer	-32,768:32,767 ($-2^{15}:2^{15}-1$)
int	System.Int32	32-bit signed integer	-2,147,483,648:2,147,483,647 ($-2^{31}:2^{31}-1$)
long	System.Int64	64-bit signed integer	-9,223,372,036,854,775,808: 9,223,372,036,854,775,807 ($-2^{63}:2^{63}-1$)
byte	System.Byte	8-bit unsigned integer	0:255 ($0:2^8-1$)
ushort	System.UInt16	16-bit unsigned integer	0:65,535 ($0:2^{16}-1$)
uint	System.UInt32	32-bit unsigned integer	0:4,294,967,295 ($0:2^{32}-1$)
ulong	System.UInt64	64-bit unsigned integer	0:18,446,744,073,709,551,615 ($0:2^{64}-1$)

- ☐ Là kiểu dữ liệu giá trị
- ☐ Có 8 kiểu số nguyên, phân biệt bởi số byte để biểu diễn và miền giá trị lưu trữ.
- ☐ Tất cả kiểu số nguyên có thể nhận biểu diễn ở nhiều loại hệ cơ số khác nhau: cơ số 10 (decimal), 16 (hexa), 8 (octal) và 2 (binary). Giá trị biểu diễn ở cơ số khác 10 thêm tiếp tố (prefix) tương ứng.

Kiểu số nguyên

```
long x = 0xab16; // số hexa, prefix là 0x hoặc 0X
int y = 01234; // số octal, prefix là 0
byte z = 0b1101; // số binary, prefix là 0b hoặc 0B
```

📖 Từ C# 7 cho phép sử dụng dấu “_” để tách các chữ số cho dễ đọc, gọi là *digit separator*

```
int bin = 0b1101_0011_1001;
int dec = 123_456;
```

📖 Khi dùng từ khóa var để khai báo biến thuộc kiểu số nguyên, mặc định C# sẽ hiểu là kiểu int. Để xác định rõ thuộc kiểu nào sử dụng ký tự viết vào cuối giá trị của biến (postfix) gồm: *U (hoặc u)* số nguyên không dấu, *L (hoặc l)* kiểu long, *UL (hoặc ul)* kiểu ulong. Đây được gọi là *integer literal*.

```
var i = 41u;
var j = 41L;
var k = 41ul;
var z = 0x41L;
```

📖 Nếu giá trị của biến vượt quá độ lớn của int, thì C# sẽ tự chọn kiểu dữ liệu phù hợp

```
var i = 4_100_000_000; // kiểu uint
var j = 5_100_000_000; // kiểu long
```

Kiểu số thực

NAME	.NET TYPE	DESCRIPTION	SIGNIFICANT FIGURES	RANGE (APPROXIMATE)
float	System.Single	32-bit, single-precision floating point	7	$\pm 1.5 \times 10^{245}$ to $\pm 3.4 \times 10^{38}$
double	System.Double	64-bit, double-precision floating point	15/16	$\pm 5.0 \times 10^{2324}$ to $\pm 1.7 \times 10^{308}$

- ☐ Là kiểu dữ liệu giá trị
- ☐ Có 2 loại: float (System.Single) và double (System.Double).
- ☐ Khi dùng từ khóa var để khai báo kiểu số thực, mặc định sẽ là *double*.
- ☐ Để chỉ định là float, dùng postfix F hoặc f.

NAME	.NET TYPE	DESCRIPTION	SIGNIFICANT FIGURES	RANGE (APPROXIMATE)
decimal	System.Decimal	128-bit, high-precision decimal notation	28	$\pm 1.0 \times 10^{228}$ to $\pm 7.9 \times 10^{28}$

- ☐ decimal Là dạng số thực đặc biệt, độ chính xác cao dùng trong tính toán tài chính; postfix là M hoặc m

Kiểu Boolean

NAME	.NET TYPE	DESCRIPTION	SIGNIFICANT FIGURES	RANGE
bool	System.Boolean	Represents true or false	NA	true or false

- Chỉ nhận 2 giá trị: **true** và **false**
- Không cho phép chuyển đổi giữa bool và số nguyên.

```
bool flag = true;
```

```
bool check = false;
```

Kiểu ký tự

NAME	.NET TYPE	VALUES
char	System.Char	Represents a single 16-bit (Unicode) character

- ❏ Dùng để biểu diễn ký tự đơn, mặc định là kiểu Unicode 16 bit.
- ❏ Khai báo giá trị đặt trong dấu nháy đơn ("), không sử dụng dấu nháy đôi " cho kiểu ký tự: `char c = 'Z';`
- ❏ Có thể sử dụng mã Unicode của ký tự: `char c = '\u005A';`
`char c = '\x005A';`
- ❏ Hoặc dùng ép kiểu char với mã decimal: `char c = (char) 90;`
- ❏ Một số ký tự đặc biệt (escape sequence)
 - \': dấu nháy đơn
 - \": dấu nháy đôi
 - \\: dấu backslash
 - \0: Null
 - \a: cảnh báo (alert)
 - \b: xóa lùi (backspace)
 - \n: dòng mới
 - \r: quay về đầu dòng
 - \t: dấu tab ngang
 - \v: dấu tab dọc

Kiểu chuỗi ký tự

NAME	.NET TYPE	DESCRIPTION
string	System.String	Unicode character string

📖 Là kiểu dữ liệu tham chiếu (reference type)

📖 Khai báo giá trị đặt trong dấu nháy đôi ("")

```
string s = "Hello";  
string s = ""; //Đây là chuỗi rỗng
```

📖 Có thể khởi tạo bằng hàm khởi tạo được cung cấp cho string.

```
char[] c = { 'H', 'e', 'l', 'l', 'o' };  
string s = new string(c);
```

📖 Có thể kết hợp ký tự đặc biệt trong chuỗi

```
string s = "Hello \r\n";
```

📖 Để gán giá trị là đường dẫn:

```
string strPath = "C:\\Study\\VS"; hoặc  
string strPath = @"C:\Study\VS";
```

Kiểu chuỗi ký tự

- Chuỗi ký tự có thể chứa biến và biểu thức, bắt đầu bằng "\$".

```
int x = 1, y = 2;
```

```
string sum = $"x = {x} y = {y} sum = {x + y}";
```

Kết quả: x = 1 y = 2 sum = 3

- Phép cộng chuỗi "+": là phép nối chuỗi.

```
string s1 = "Hello", s2 = "world";
```

```
string str = s1 + " " + s2;
```

Kết quả: str = "Hello world";

- Chuỗi trong C# có thể coi là mảng một chiều; có thể truy xuất từng phần tử trong chuỗi như trong mảng. VD: `str[1]` là 'e'

- Không cho phép thay đổi giá trị ký tự trong chuỗi theo chỉ số.

VD: `str[2] = 'K';` //Lỗi

- Lớp String cung cấp một số đặc tính và phương thức để làm việc với chuỗi.

- `Length`: trả về số ký tự trong chuỗi.

- `bool Contains(string value)`: kiểm tra có chuỗi con value không

Kiểu chuỗi ký tự

- `bool EndsWith(string value)`: kiểm tra chuỗi con value có nằm cuối chuỗi không
- `bool StartsWith(string value)`: kiểm tra chuỗi con value có nằm đầu chuỗi không.
- `bool Equals(string value)`: so sánh chuỗi với value.
- `string ToLower()`: tạo ra bản sao của chuỗi và chuyển chuỗi thành chữ thường.
- `string ToUpper()`: tạo ra bản sao của chuỗi và chuyển chuỗi thành chữ hoa.
- `string Trim()`: tạo ra bản sao của chuỗi và cắt bỏ hết khoảng trắng ở đầu và cuối chuỗi: ký tự space, tab, \r, \n.
- Tương tự `TrimStart()` cắt bỏ khoảng trắng đầu chuỗi; `TrimEnd()` cắt bỏ khoảng trắng cuối chuỗi.
- `Trim()`, `TrimStart()` và `TrimEnd()` còn có thể cắt bỏ những ký tự theo yêu cầu: `str.Trim(new[] { 'l', 'w' });`

Kiểu chuỗi ký tự

- `int IndexOf(string value)`: xác định vị trí chuỗi con value; nếu value xuất hiện nhiều lần, sẽ trả về vị trí gặp đầu tiên.
- `int LastIndexOf(string value)`: xác định vị trí chuỗi con value; nếu value xuất hiện nhiều lần, sẽ trả về vị trí gặp cuối cùng.
- `string Insert(int starIndex, string value)`: tạo ra bản sao của chuỗi và thêm chuỗi con value vào vị trí starIndex.
- `string Remove(int starIndex, int count)`: tạo ra bản sao của chuỗi nhưng bỏ đi count ký tự tính từ vị trí starIndex.
- `string Replace(string oldValue, string newValue)`: tạo ra bản sao của chuỗi nhưng thay thế chuỗi con bằng chuỗi con khác.
- `string[] Split(param char[] separator)`: tách chuỗi thành nhiều chuỗi con bằng ký tự đánh dấu.

```
string[] kq = "Ngày mai nghỉ học".Split(new[] { ' ' });
```

- `char[] ToCharArray()`: chuyển đổi chuỗi thành mảng char[]
- `char[] ToCharArray(int starIndex, int length)`: chuyển đổi một phần chuỗi thành mảng tính từ vị trí starIndex tới length ký tự

Kiểu chuỗi ký tự

- ☐ Một số phương thức static của string: gọi trực tiếp từ lớp *string*
 - `static int Compare(string strA, string strB)`: so sánh 2 chuỗi strA và strB; bằng trả về 0, strA > strB trả về 1, strA < strB trả về -1.
`string.Compare("World", "world"); KQ = 1`
 - `static bool Equals(string a, string b)`: xác định chuỗi a có bằng chuỗi b không.
 - `static string Concat(params string[] values)`: ghép nhiều chuỗi con thành chuỗi lớn.
 - `static string Join(string separator, params string[] values)`: giống Concat, nhưng sẽ chèn thêm chuỗi separator vào giữa các chuỗi con.
 - `static string Copy(string str)`: tạo bản sao của chuỗi.
 - `static bool IsNullOrEmpty(string value)`: kiểm tra chuỗi value là null hay rỗng.
 - `string Substring (int startIndex)`: trả về chuỗi mới từ vị trí startIndex.

Kiểu chuỗi ký tự

- `static string Format(string format, Object arg0)`: tạo ra chuỗi từ giá trị của biến và các định dạng

```
string str = string.Format("Tổng của {0} + {1} = {2}", x, y, x + y);
```

KQ = Tổng của 2 + 1 = 3

Các kiểu định dạng với `string.Format()`:

- Định dạng số gồm 3 phần: format, precision và alignment

- *Format*: với các kiểu định dạng: *Currency* (C/c), *Decimal* (D/d), *Fixed point* (F/f), *General* (G/g), *Hexadecimal* (X/x), *Number* (N/n), *Percent* (P/p), *Round-trip* (R/r), *Scientific* (E/e)

```
string.Format("Tiền: {0:C}", 1000); KQ = "Tiền: $1,000.00"
```

- *Precision*: mô tả độ chính xác (phụ thuộc vào loại số và định dạng) của giá trị (số) được in ra.

```
string.Format("Tiền: {0:c3}", 1000); KQ = "Tiền: $1,000.000"
```

- *Alignment*: chỉ định số lượng ký tự tối thiểu để in giá trị

```
string.Format("{0, 10}VND", 1000); KQ= "          1000VND"
```

```
string.Format("{0, -10}VND", 1000); KQ= "1000          VND"
```

Kiểu chuỗi ký tự

○ Định dạng thời gian: `DateTime`

- `d`: thông tin về ngày ở dạng ngắn gọn
- `D`: thông tin về ngày ở dạng đầy đủ.
- `t`: thông tin về thời gian ở dạng ngắn gọn.
- `T`: thông tin về thời gian ở dạng đầy đủ.
- Đưa ra định dạng ngày tháng riêng.

```
> var day = new DateTime(2020, 2, 29);  
> string.Format("Ngày: {0}", day)  
"Ngày: 29/02/2020 12:00:00 AM"  
> string.Format("Ngày: {0:d}", day)  
"Ngày: 29/02/2020"  
> string.Format("Ngày: {0:D}", day)  
"Ngày: Saturday, February 29, 2020"  
> string.Format("Ngày: {0:t}", day)  
"Ngày: 12:00 AM"  
> string.Format("Ngày: {0,10:T}", day)  
"Ngày: 12:00:00 AM"  
> string.Format("Ngày: {0:dd/MM/yyyy}", day)  
"Ngày: 29/02/2020"  
> string.Format("Ngày: {0:d-MM-yyyy}", day)  
"Ngày: 29-02-2020"  
> string.Format("Ngày: {0:dd-MMM-yy}", day)  
"Ngày: 29-Feb-20"
```

Kiểu object

NAME	.NET TYPE	DESCRIPTION
object	System.Object	The root type. All other types (including value types) are derived from object.

- ❏ Là kiểu dữ liệu gốc cho mọi kiểu dữ liệu khác (root type)
- ❏ Là kiểu dữ liệu tham chiếu.
- ❏ Phương thức quan trọng trong kiểu object: **ToString()** và có mặt trong tất cả các kiểu dữ liệu trong C#. Được dùng để chuyển đổi giá trị của kiểu tương ứng về chuỗi ký tự.

```
int x = 9876;  
string s = x.ToString();
```

Kiểu nullable

- ☐ Các biến kiểu tham chiếu có một giá trị đặc biệt: `null`.







```
string s = null;
```

- ☐ Tất cả các biến kiểu tham chiếu khi khai báo đều nhận giá trị mặc định là *null* và bắt buộc khởi tạo trước khi sử dụng.
- ☐ Mọi thao tác trên biến kiểu tham chiếu có giá trị null đều sẽ báo lỗi.
- ☐ Biến kiểu giá trị không thể nhận giá trị *null*.
- ☐ Để biến kiểu giá trị nhận giá trị null, thêm ký tự “?” (nullable type) vào sau tên kiểu giá trị: `int? sum = null;`



Toán tử

Toán tử (operator)

-  Toán tử số học
-  Toán tử trên bit
-  Toán tử logic
-  Toán tử quan hệ
-  Toán tử gán
-  Toán tử khác

Toán tử số học

Toán tử	Ý nghĩa	Ví dụ
+	Số dương	+x
-	Số âm	-x
++	Tăng sau	x++
--	Giảm sau	x--
++	Tăng trước	++x
--	Giảm trước	--x
+	Cộng	x + y
-	Trừ	x - y
*	Nhân	x * y
/	Chia	x / y
%	Chia lấy dư	x % y

```
> int x = 10;  
> int y = x++;  
> x  
11  
> y  
10
```

```
> int x = 10;  
> int y = ++x;  
> x  
11  
> y  
11
```

Toán tử trên bit

Toán tử	Ý nghĩa	Ví dụ
~	Phép bù (bitwise negation)	~x
&	Phép và (bitwise AND)	x & y
	Phép hoặc (bitwise OR)	x y
^	Phép bitwise XOR	x ^ y
<<	Dịch trái (shift left)	x << y
>>	Dịch phải (shift right)	x >> y

```
> int x = 0b1010; //x =10;
> int y = 0b1001; //y = 9
> x & y
8
> x | y
11
> x ^ y
3
> ~x
-11
> x << 2
40
> x >> 2
2
```

Toán tử logic và toán tử quan hệ

Toán tử	Ý nghĩa	Ví dụ
&&	Và (and)	$x \&\& y$
	Hoặc (or)	$x y$
!	Phủ định (Not)	$!x$
==	So sánh bằng	$x == y$
!=	So sánh không bằng	$x != y$
>	Lớn hơn	$x > y$
>=	Lớn hơn hoặc bằng	$x >= y$
<	Nhỏ hơn	$x < y$
<=	Nhỏ hơn hoặc bằng	$x <= y$

Toán tử gán

Toán tử	Ý nghĩa	Ví dụ
=	Phép gán	$x = 10;$
+=	Phép cộng rồi gán	$x += 5;$
-=	Phép trừ rồi gán	$x -= 5;$
*=	Phép nhân rồi gán	$x *= 5;$
/=	Phép chia rồi gán	$x /= 5;$
%=	Phép chia lấy dư rồi gán	$x \% = y$

Một số toán tử khác

- ▣ `sizeof()`: trả về kích cỡ một kiểu dữ liệu. VD: `sizeof(int)`, kết quả bằng 4.
- ▣ `typeof()`: trả về object chứa thông tin về kiểu dữ liệu. VD: `typeof(string)`, kết quả: *System.String*.
- ▣ `new`: cấp phát vùng nhớ mới cho kiểu dữ liệu tham chiếu.
- ▣ Toán tử ép kiểu – type casting: `(): (kiểu mới) <giá trị>;` Nếu kết quả ép kiểu không thành công sẽ phát ra ngoại lệ.
- ▣ `as`: phép ép kiểu cho kiểu tham chiếu và không phát sinh ngoại lệ vì nếu ép không thành công thì sẽ trả về giá trị null.
- ▣ `is`: để kiểm tra kiểu của một đối tượng. Kết quả trả về true hoặc false.

```
> object o = "Hello";  
> o is string  
true  
> o is int  
false|
```

Một số toán tử khác

📖 Toán tử điều kiện “? :” :

<biểu thức điều kiện> ? <giá trị 1> : <giá trị 2>




Biểu thức điều kiện đúng: trả về giá trị 1; sai: trả về giá trị 2

```
int x = 10, y = 20;  
int z = (x > y) ? x : y;
```



Cấu trúc điều khiển

Cấu trúc điều khiển

-  Cấu trúc rẽ nhánh if else
-  Cấu trúc rẽ nhánh switch - case
-  Cấu trúc lặp

Cấu trúc rẽ nhánh if else

Dạng 1:

```
if (<biểu thức điều kiện>){  
    <câu lệnh>;  
}
```

Dạng 2:

```
if (<biểu thức điều kiện>){  
    <câu lệnh>;  
}  
else{  
    <câu lệnh>  
}
```

Dạng 3:

```
if (<biểu thức điều kiện>){  
    <câu lệnh>;  
}  
else if (<biểu thức điều kiện>){  
    <câu lệnh>  
}  
else{  
    <câu lệnh>  
}
```

Cấu trúc rẽ nhánh switch-case

Cú pháp:

```
switch (<biểu thức điều kiện>)
```

```
{
```

```
    case <giá trị 1>:
```

```
        <khối lệnh>;
```

```
        break;
```

```
    case <giá trị 2>:
```

```
        <khối lệnh>;
```

```
        break;
```

```
    case <giá trị 3>:
```

```
        <khối lệnh>;
```

```
        break;
```

```
    [default:
```

```
        <khối lệnh>;
```

```
        break;]
```

```
}
```

Là giá trị hằng

Để thoát khỏi switch

Không bắt buộc phải có

Cấu trúc lặp

📖 Có 4 cấu trúc lặp: while, do - while, for, foreach.

📖 Cấu trúc **while**:

```
while (<biểu thức điều kiện>) {[danh sách câu lệnh]}
```

📖 Cấu trúc **do - while**:

```
do {[danh sách câu lệnh]} while (<biểu thức điều kiện>;
```

📖 Cấu trúc **for**:

```
for ([<khởi tạo biến đếm>; [<điều kiện lặp>]; [<bước lặp>])  
{  
    [danh sách câu lệnh]  
}
```

📖 Có thể dùng lệnh **break** hoặc **continue** để điều khiển hoạt động của vòng lặp.

- Break: khi gặp break sẽ thoát khỏi vòng lặp. tất cả các lệnh sau break sẽ không được thực hiện.
- Continue: dừng lệnh lặp hiện tại, tất cả các lệnh sau continue sẽ không được thực hiện, vòng lặp sẽ tiếp tục ở chu kỳ lặp tiếp theo.

Cấu trúc lặp

▣ Cấu trúc lặp **foreach**: dùng để duyệt phần tử của mảng hoặc tập hợp.

```
foreach (<kiểu dữ liệu> <biến tạm> in <mảng hoặc tập hợp>)  
{  
    [danh sách câu lệnh]  
}
```

▣ Tự động duyệt qua các phần tử của mảng ở mỗi lần lặp.

▣ Khi duyệt phần tử nào thì giá trị của phần tử đó được lưu trong biến tạm

▣ Trong vòng lặp sử dụng luôn biến tạm để lấy giá trị của phần tử đang xét.

▣ Khi duyệt qua hết phần tử vòng lặp sẽ kết thúc.

▣ Một số lưu ý khi dùng foreach:

- Nếu mảng không có phần tử thì foreach không thực thi.
- Duyệt từ phần tử đầu mảng cho đến cuối, không có chiều ngược lại.
- Không được phép thay đổi giá trị của biến tạm cho nên cũng không thể thay đổi giá trị phần tử trong mảng bằng biến tạm.



Mảng

Mảng (array)

- 📖 Mảng (array): là tập hợp dữ liệu có cùng kiểu dữ liệu.
 - Dữ liệu đơn lẻ được gọi là phần tử.
 - Phần tử được đánh chỉ số (index), sắp xếp liên nhau
 - Kiểu dữ liệu của phần tử là *kiểu cơ sở* của mảng, và là kiểu dữ liệu hệ thống hoặc do người dùng định nghĩa.
 - Chỉ số bắt đầu của mảng là 0.
 - Mảng là object thuộc lớp *System.Array*
- 📖 Các loại mảng trong C#:
 - Mảng một chiều
 - Mảng nhiều chiều
 - Mảng răng cưa (jagged)

Mảng một chiều

📖 Cú pháp khai báo: `<kiểu dữ liệu>[] <tên mảng>;`

📖 VD: `string[] cities; int[] numbers;`

📖 Dùng từ khóa `new` để khởi tạo mảng.

○ Cách 1: cấp phát số lượng phần tử cần dùng

```
string[] cities = new string[20];
```

○ Cách 2: cung cấp giá trị các phần tử của mảng lúc khởi tạo

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

○ Cách 3: cung cấp giá trị các phần tử của mảng lúc khai báo, không dùng được với var.

```
int[] numbers = { 1, 2, 3 };
```

📖 Mảng một khi được khởi tạo thì không thể thay đổi được số lượng phần tử.

📖 Dùng vòng lặp foreach để duyệt mảng.

Mảng một chiều

Truy xuất phần tử của mảng:

- Bằng chỉ số index: <tên mảng>[chỉ số]

```
cities[1] = giá trị;
```

```
biến = cities[2];
```

- Bằng phương thức: <tên mảng>.SetValue(<giá trị>, <chỉ số>);
<tên mảng>.GetValue(<chỉ số>);

```
cities.SetValue('ABC', 2);
```

```
biến = cities.GetValue(2);
```

Một số thuộc tính và phương thức của mảng một chiều

Thuộc tính hoặc phương thức	Ý nghĩa
Length/LongLength	Trả về số phần tử tối đa của mảng theo kiểu int/long
GetLength(<dòng số>)/ GetLongLength(<dòng số>)	Phương thức trả về số phần tử tối đa trong một dòng của mảng theo kiểu int/long . Mảng 1 chiều <i>dòng số</i> = 0
Sort()	Phương thức sắp xếp mảng theo thứ tự
Clear()	Phương thức xóa hết giá trị phần tử trong mảng.
Copy()	Phương thức sao chép giá trị của mảng ra một vùng nhớ mới.
Reverse()	Phương thức đảo ngược thứ tự của mảng

Mảng nhiều chiều

- 📖 C# hỗ trợ mảng tối đa 32 chiều.
- 📖 Chủ yếu sử dụng mảng 2 chiều.
- 📖 Mảng 2 chiều: như là một bảng có n dòng và m cột:
 - Có những đặc trưng cơ bản của một mảng bình thường
 - Các phần tử trong mảng 2 chiều được truy xuất thông qua 2 chỉ số (số dòng và số cột).
 - Chỉ số đầu tiên của dòng và cột là 0.
- 📖 Cú pháp khai báo: `<kiểu dữ liệu>[,] <tên mảng>;`
- 📖 VD: `int[,] matrix;`
- 📖 Dùng từ khóa `new` để khởi tạo mảng, khi khởi tạo cần xác định số dòng và số cột tối đa của mảng.
 - Cách 1: cấp phát số lượng phần tử cần dùng

```
int[,] matrix = new int[2,2]{  
    {11, 12},  
    {21, 22}  
};
```

Mảng nhiều chiều

- Cách 2: cung cấp giá trị phần tử của mảng lúc khởi tạo

```
int[,] matrix = new int[  
{  
    {11, 12},  
    {21, 22}  
};
```

- Cách 3: cung cấp giá trị phần tử của mảng không cần **new**

```
int[,] matrix =  
{  
    {11, 12},  
    {21, 22}  
};
```

📖 Truy xuất phần tử của mảng 2 chiều: `matrix[i, j]`

📖 Sử dụng 2 vòng lặp for lồng nhau để duyệt mảng 2 chiều

```
for (int i = 0; i < matrix.GetLength(0); i++)  
{  
    for (int j = 0; j < matrix.GetLength(1); j++)  
    {  
        // Câu lệnh  
    }  
}
```

Mảng nhiều chiều

📖 Một số thuộc tính và phương thức của mảng hai chiều

Thuộc tính hoặc phương thức	Ý nghĩa
Length/LongLength	Trả về số phần tử tối đa của mảng (bằng tích số dòng và số cột) theo kiểu int/long
GetLength(<dòng số>)/ GetLongLength(<dòng số>)	Phương thức trả về số phần tử tối đa trong một dòng theo kiểu int/long . Dòng đầu tiên thì <i>dòng số</i> = 0
Rank	Trả về số chiều của mảng
Clone()	Phương thức sao chép giá trị của mảng ra một vùng nhớ mới.

Mảng răng cưa (Jagged array)

- ❏ Mảng răng cưa (jagged array) còn gọi là mảng của mảng; đó là mảng một chiều và mỗi phần tử trong mảng lại là một mảng.
- ❏ Mỗi phần tử của mảng cùng chung kiểu cơ sở và có thể khác kích thước.
- ❏ Cú pháp khai báo: `<kiểu dữ liệu>[][] <tên mảng>;`
- ❏ VD: `int[][] jagged;`
- ❏ Khởi tạo mảng: cấp phát số dòng trước; sau đó, ứng với mỗi dòng cấp phát số cột sau.

```
int[][] jagged = new int [2][];  
jagged[0] = new int [4];  
jagged[1] = new int [2];
```

- ❏ Khởi tạo giá trị: tương tự cho mảng một chiều và mảng hai chiều.

```
int[][] jagged =  
{  
    new int[] {1, 2, 3, 4},  
    new int[] {5, 6}  
};
```

Mảng răng cưa (Jagged array)

Truy xuất phần tử mảng: <tên mảng>[i][j]

- [i] chỉ vị trí phần tử trong mảng 1 chiều.
- [j] chỉ vị trí phần tử trong mảng 1 chiều được cấp phát sau. Nếu mảng là mảng 2 chiều thì sẽ thay bằng cách truy xuất của mảng 2 chiều.
- Để duyệt mảng jagged dùng 1 vòng lặp for để duyệt mảng 1 chiều lớn, còn mảng trong thì duyệt mảng tùy thuộc và mảng 1 chiều hoặc 2 chiều.

```
for (int i = 0; i < jagged.Length; i++)  
{  
    //Phụ thuộc vào mảng trong là mảng nào.  
}
```



Kiểu liệt kê

Kiểu liệt kê (enum)

📖 Kiểu dữ liệu liệt kê – enum (enumeration): là kiểu dữ liệu do người dùng định nghĩa:

- Chứa hữu hạn một danh sách hằng số.
- Mỗi hằng số được đặt tên và giá trị là số nguyên.

📖 Khai báo:

```
enum <tên enum>{ <danh sách các hằng> };
```

📖 Đây là một kiểu liệt kê với các hằng RED, BLUE, GREEN

```
enum Color  
{  
    RED,  
    BLUE,  
    GREEN  
}
```

- Các giá trị số nguyên cho hằng theo thứ tự tăng dần, bắt đầu từ 0.

Kiểu liệt kê (enum)

- Tương tự như khai báo:

```
public const int RED = 0;  
public const int BLUE = 1;  
public const int GREEN = 2;
```

- 📖 Có thể quy định giá trị cho từng hằng.

```
enum Color  
{  
    RED = 2,  
    BLUE = 5,  
    GREEN = 7  
}
```

- 📖 Mặc định giá trị hằng là *int*, nhưng có thể đổi sang kiểu khác.

```
enum Color : byte  
{  
    RED = 2,  
    BLUE = 5,  
    GREEN = 7  
}
```


Kiểu liệt kê (enum)

- Truy xuất giá trị của enum qua toán tử “.”: `Color.RED`;
- Giá trị của hằng trong enum là kiểu số nguyên nhưng không thể so sánh trực tiếp với một số nguyên được, mà phải ép kiểu.

```
if (Color.RED == 3) //Sai
if ((int)Color.RED == 3) //Đúng
```

- Có thể ép kiểu một số nguyên sang kiểu liệt kê.

```
Color red = (int) 3;
```

- Cách duyệt enum bằng vòng foreach

```
foreach (var color in Enum.GetNames(typeof(Color)))
{
    Console.WriteLine($"Màu: {color}");
}
```



Namespace

Namespace

- 📖 Namespace: được dùng để quản lý code, chứa đựng các struct, class, interface, enum, delegate hoặc namespace khác.
- 📖 Ví dụ: namespace **System**.
- 📖 Quy tắc đặt tên namespace:
 - Bắt đầu bằng ký tự chữ hoặc ký tự gạch chân “_”.
 - Chỉ chứa chữ cái, chữ số và gạch chân.
 - Phân biệt chữ hoa, thường.
- 📖 Quy ước đặt tên (không bắt buộc): namespace gốc nên trùng với tên project.
- 📖 Cách sử dụng các lớp trong namespace:
`<tên namespace>.<tên thành viên trong namespace>`
Tên thành viên trong namespace: có thể là class, struct, enum...

Namespace

VD: `System.Console.WriteLine();`

- `System`: tên namespace
- `Console`: tên class trong namespace
- `WriteLine`: phương thức trong class `Console`

Cấu trúc `using` cho phép sử dụng tất cả các thành viên trong namespace mà không cần gọi tên namespace

```
using System;

namespace FirstApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine();
        }
    }
}
```

Namespace

- ❏ Cấu trúc `using static` cho phép sử dụng tất cả các thành viên static của class, struct mà không cần gọi tên class, struct

```
using System;
using static System.Console;
namespace FirstApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            WriteLine();
        }
    }
}
```