TEDU-46: C# căn bản đến nâng cao

Contents

[Bài 1: Giới thiệu về ngôn ngữ C# và .NET Platform 1](#_Toc112576149)

[Giới thiệu khoá học 1](#_Toc112576150)

[Giới thiệu .NET 1](#_Toc112576151)

[Lịch sử phát triển 2](#_Toc112576152)

[Ngôn ngữ C# 2](#_Toc112576153)

[Bài 2: Cài đặt Visual Studio Community 2022 và giới thiệu giao diện 3](#_Toc112576154)

[Cài đặt môi trường 3](#_Toc112576155)

[Giới thiệu giao diện Visual Studio 3](#_Toc112576156)

[Bài 3: Viết chương trình C# đầu tiên 3](#_Toc112576157)

[Bài 4: Tổng quan về khái niệm kiểu dữ liệu và biến 3](#_Toc112576158)

[Khái niệm về biến 3](#_Toc112576159)

[Khái niệm kiểu dữ liệu 4](#_Toc112576160)

[Cách khai báo biến 5](#_Toc112576161)

[Cách khai báo biến với các kiểu dữ liệu tiêu biểu 6](#_Toc112576162)

[Bài 5: Tìm hiểu về các kiểu dữ liệu hay dùng nhất. 6](#_Toc112576163)

[Khai báo giá trị với giá trị mặc định 6](#_Toc112576164)

[Khái niệm CLS và CTS 7](#_Toc112576165)

[Bảng tổng hợp một số kiểu dữ liệu cơ bản 8](#_Toc112576166)

[Kiểu string 10](#_Toc112576167)

[Lưu ý 10](#_Toc112576168)

[Bài 6: Thực hành về kiểu int, float, double 10](#_Toc112576169)

[Bài 7: Kiểu dữ liệu String 11](#_Toc112576170)

[Bài 8: Quy tắc code chuẩn 11](#_Toc112576171)

[Bài 9: Kiểu tham trị và kiểu tham chiếu 11](#_Toc112576172)

[Kiểu tham trị (Value Type) 12](#_Toc112576173)

[Kiểu tham chiếu 13](#_Toc112576174)

[Bộ nhớ Stack và Heap 13](#_Toc112576175)

[So sánh giữa Stack và Heap 13](#_Toc112576176)

[So sánh cấp phát tĩnh và cấp phát động 13](#_Toc112576177)

[Tóm lại 14](#_Toc112576178)

[Bài 10: Nhập xuất cơ bản với màn hình Console 15](#_Toc112576179)

[Ứng dụng Console là gì? 15](#_Toc112576180)

[Xuất dữ liệu với Console 15](#_Toc112576181)

[Nhập dữ liệu với Console 20](#_Toc112576182)

[Bài 11: Chuyển đổi kiểu dữ liệu (Type Casting) 23](#_Toc112576183)

[Implicit Casting (chuyển đổi ngầm định tự động) 24](#_Toc112576184)

[Explicit Casting (chuyển đổi tường minh bằng tay) 24](#_Toc112576185)

[Khác nhau giữa casting, parsing và converting 24](#_Toc112576186)

[Casting: là chuyển 1 giá trị từ kiểu này sang kiểu khác hoặc xuất ra lỗi 24](#_Toc112576187)

[Conversion: Là cố chuyển một kiểu đối tượng sang kiểu khác, ít lỗi hơn nhưng chậm hơn 25](#_Toc112576188)

[Parsing: Là cố chuyển một chuỗi sang một kiểu nguyên thuỷ 25](#_Toc112576189)

[Khác nhau giữa Parse và Convert 25](#_Toc112576190)

[TryParse() 25](#_Toc112576191)

[Từ khoá mới 25](#_Toc112576192)

[Từ khoá is: Sử dụng để kiểm tra nếu một giá trị cụ thể là một kiểu cụ thể 25](#_Toc112576193)

[Từ khoá as: Sử dụng để chuyển một object từ một kiểu sang một kiểu khác 25](#_Toc112576194)

[Từ khoá typeof: Trả về kiểu của một đối tượng 26](#_Toc112576195)

[Tổng kết 26](#_Toc112576196)

[Bài 12: Hằng số (const) 27](#_Toc112576197)

[Bài 13: Bài tập tổng kết chương 1 28](#_Toc112576198)

[Bài 14: Method 28](#_Toc112576199)

[Phương thức là gì? 28](#_Toc112576200)

[Cú pháp khai báo 28](#_Toc112576201)

[Void method 30](#_Toc112576202)

[Ví dụ thực hành 30](#_Toc112576203)

[Bài 15: Exception Handler 30](#_Toc112576204)

[Khái niệm Exception 30](#_Toc112576205)

[Từ khoá 30](#_Toc112576206)

[Cú pháp xử lý 31](#_Toc112576207)

[Các lớp Exception của hệ thống 32](#_Toc112576208)

[Ví dụ 32](#_Toc112576209)

[Bài 16: Toán tử (operators) 32](#_Toc112576210)

[Bài 17: Cấu trúc điều khiển if else 33](#_Toc112576211)

[Bài 18: Cấu trúc điều khiển Swicth case 34](#_Toc112576212)

[Bài 19: Vòng lặp (loop) 35](#_Toc112576213)

[Các kiểu vòng lặp 35](#_Toc112576214)

[Vòng lặp while 36](#_Toc112576215)

[Vòng lặp do…while 37](#_Toc112576216)

[Vòng lặp for 39](#_Toc112576217)

[Vòng lặp foreach 40](#_Toc112576218)

[Câu lệnh break 42](#_Toc112576219)

[Câu lệnh continue 44](#_Toc112576220)

[Bài 20: Bài tập kết thúc chương 2 46](#_Toc112576221)

[Bài 21: Lập trình hướng đối tượng 46](#_Toc112576222)

[Giới thiệu về Class và Object 46](#_Toc112576223)

[Sử dụng Constructor (1 constructor và nhiều constructor) 47](#_Toc112576224)

[Phạm vi truy cập (Access Modifiers) 47](#_Toc112576225)

[Tạo và sử dụng đối tượng 48](#_Toc112576226)

[Từ khoá this 48](#_Toc112576227)

[Thuộc tính (Properties) 48](#_Toc112576228)

[Trường dữ liệu của lớp 48](#_Toc112576229)

[Thuộc tính, bộ truy cập accessor setter/getter 48](#_Toc112576230)

[Tìm hiểu tính đóng gói lập trình hướng đối tượng 50](#_Toc112576231)

[Bài 22: Sử dụng mảng (Arrays) 50](#_Toc112576232)

[Giới thiệu về mảng 50](#_Toc112576233)

[Khai báo và khởi tạo mảng 51](#_Toc112576234)

[Khai báo 51](#_Toc112576235)

[Khai báo và khởi tạo 51](#_Toc112576236)

[Gán giá trị cho một mảng 52](#_Toc112576237)

[Thực hành tạo mảng và truy xuất giá trị 52](#_Toc112576238)

[Lặp qua mảng 52](#_Toc112576239)

[Mảng 2 chiều 52](#_Toc112576240)

[Jagged Array 53](#_Toc112576241)

[Bài 23: Tổng quan về Generic và Non-Generic Collection 53](#_Toc112576242)

[Bài 24: Các loại Collection 54](#_Toc112576243)

[Lists 54](#_Toc112576244)

[Dictionaries 54](#_Toc112576245)

[Queue & Stack 54](#_Toc112576246)

[Bài 25: Cách debug ứng dụng C# 55](#_Toc112576247)

[Bài 26: Tính chất kế thừa (Inheritance) 55](#_Toc112576248)

[Giới thiệu về tính chất kế thừa 55](#_Toc112576249)

[Ví dụ demo 55](#_Toc112576250)

[Từ khoá virtual và override 55](#_Toc112576251)

[Bài 27: Tìm hiểu về Interface 55](#_Toc112576252)

[Bài 28: Giới thiệu về IEnumerator và IEnumberable 55](#_Toc112576253)

[Bài 29: Tính đa hình (Polymorphism) 55](#_Toc112576254)

[Giới thiệu tính đa hình 55](#_Toc112576255)

[Từ khoá sealed 55](#_Toc112576256)

[Abstract class 55](#_Toc112576257)

[Interface và abstract class 55](#_Toc112576258)

[Ví dụ về đọc ghi 1 file 55](#_Toc112576259)

[Bài 30: Tính trừu tượng (Abstraction) 55](#_Toc112576260)

[Bài 31: Sử dụng kiểu tập hợp (Enum) 55](#_Toc112576261)

[Bài 32: Sử dụng Math class 55](#_Toc112576262)

[Bài 33: Sử dụng DateTime 55](#_Toc112576263)

[Bài 34: Kiểu Nullable 55](#_Toc112576264)

[Bài 35: Hiểu và sử dụng Lamda Expression 55](#_Toc112576265)

[Bài 36: Sử dụng Multithreads 55](#_Toc112576266)

[Bài 36: Tổng kết khoá học 55](#_Toc112576267)

[Tổng kết những vấn đề đã học 55](#_Toc112576268)

[Các vấn đề cần tìm hiểu thêm (chưa nằm trong khoá học) 55](#_Toc112576269)

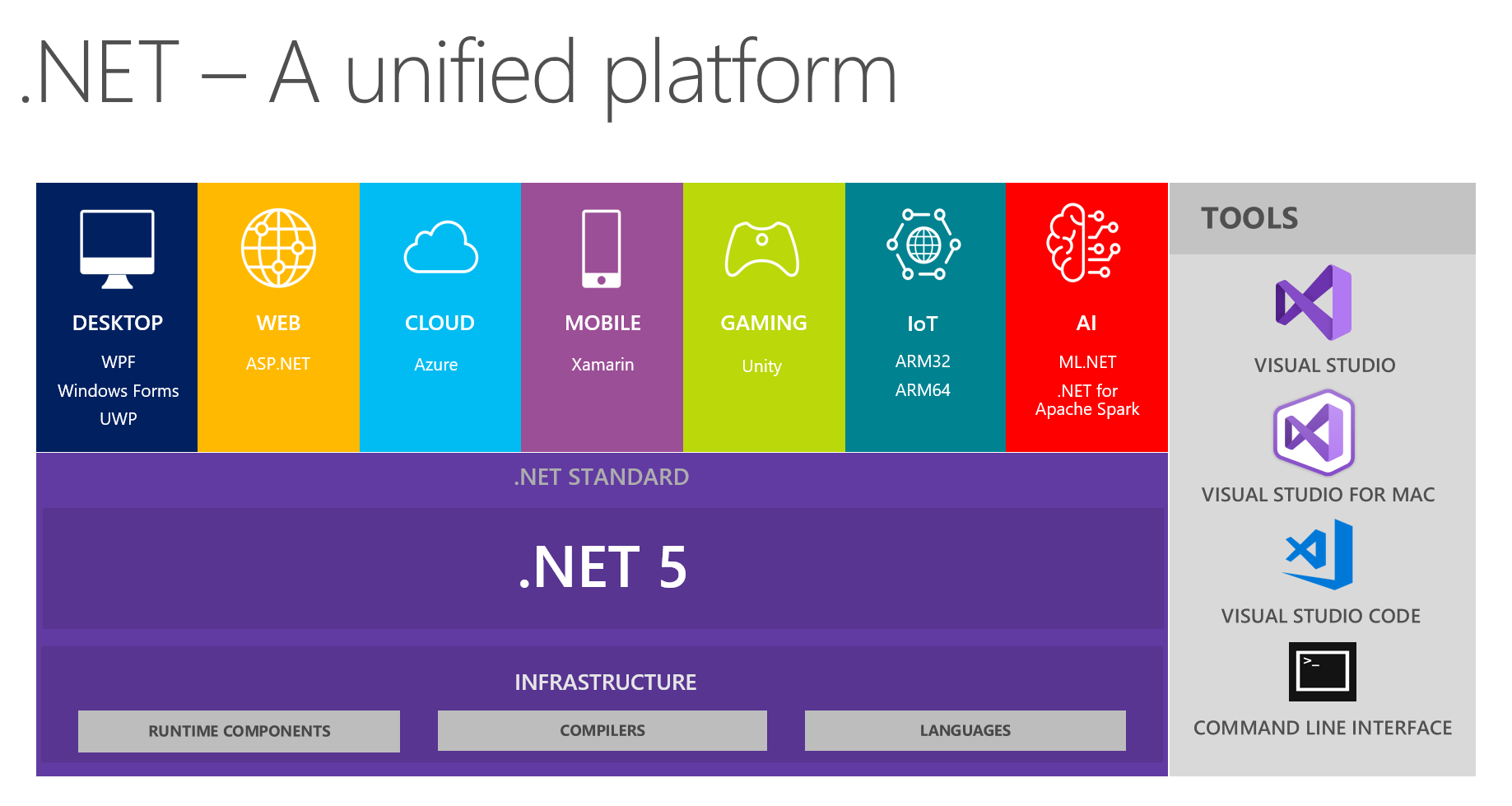
# Bài 1: Giới thiệu về ngôn ngữ C# và .NET Platform

## Giới thiệu khoá học

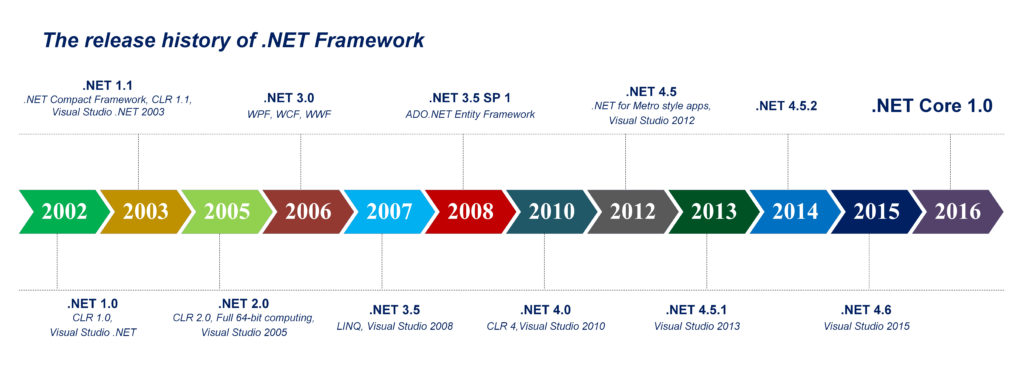
* Đây là khoá học hướng dẫn cho các bạn chưa biết gì về C# muốn theo lập trình C# và .NET.
* Trong quá trình học các bạn nhớ like và share video nếu thấy hữu ích. Các bạn cũng có thể comment để hỏi bất cứ vấn đề gì chưa rõ.
* Khoá học này làm nền tảng cho các khoá học khác tại TEDU.COM.VN.
* Sau khoá học này mình sẽ ra mắt khoá học nâng cao hơn về C# ví dụ như WPF.
* Các bạn có thể join vào link Group facebook hoặc Discord của TEDU ở link mô tả Video.
* Các bạn có thể download tài liệu này tại lớp học khoá học TEDU-46 (Lập trình C# toàn tập cho người mới bắt đầu) trên trang web chính thức TEDU.COM.VN

## Giới thiệu .NET

* Giới thiệu về .NET Platform là một nền tảng thống nhất phát triển nhiều loại ứng dụng từ Mobile, Desktop cho đến Web được phát triển bởi Microsoft.
* .NET Framework được Microsoft đưa ra chính thức từ năm 2002. .NET Framework chỉ hoạt động trên Windows. Những nền tảng ứng dụng như WPF, Winforms, ASP.NET (1-4) hoạt động dựa trên .NET Framework.
* Mono là phiên bản cộng đồng nhằm mang .NET đến những nền tảng ngoài Windows. Mono được phát triển chủ yếu nhằm xây dựng những ứng dụng với giao diện người dùng và được sử dụng rất rộng rãi: Unity Game, Xamarin…
* Cho đến năm 2013, Microsoft định hướng đi đa nền tảng và phát triển .NET core. .NET core hiện được sử dụng trong các ứng dụng Universal Windows platform và ASP.NET Core. Từ đây, C# có thể được sử dụng để phát triển các loại ứng dụng đa nền tảng trên các hệ điều hành khác nhau (Windows, Linux, MacOS,)

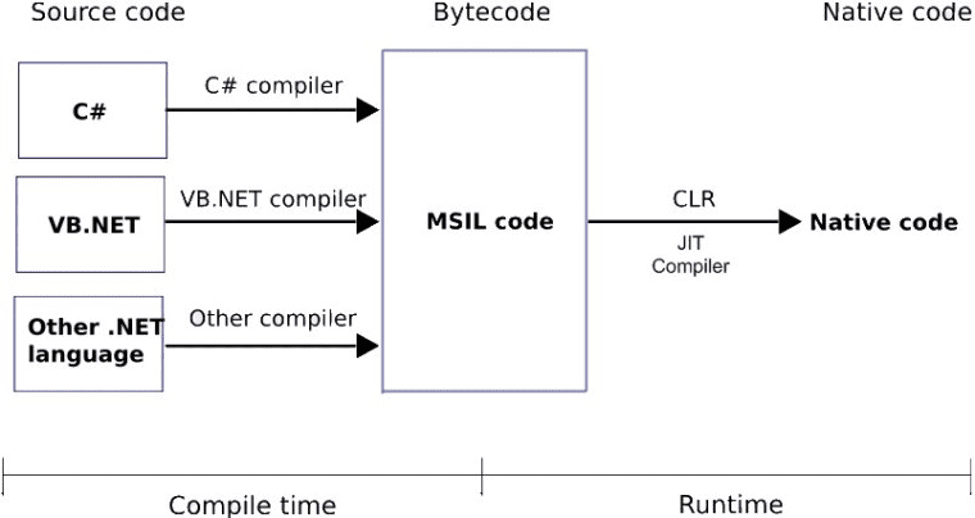


## Lịch sử phát triển



## Ngôn ngữ C#

* C# (hay C sharp) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000.
* C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.



* Lịch sử phiên bản C#: [The history of C# - C# Guide | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history)

# Bài 2: Cài đặt Visual Studio Community 2022 và giới thiệu giao diện

## Cài đặt môi trường

1. Download Visual Studio 2022 tại: <https://visualstudio.microsoft.com/> sau đó cài đặt trực tiếp trên máy.
2. Download .NET 6 SDK tại: <https://dotnet.microsoft.com/en-us/download/dotnet/6.0>
3. Test thử xem đã cài đặt thành công hay chưa? Bằng cách mở cửa sổ CMD gõ **dotnet –list-sdks**

## Giới thiệu giao diện Visual Studio

* Màn hình khởi động
* Các cửa số cần thiết
* Tuỳ chỉnh giao diện

# Bài 3: Viết chương trình C# đầu tiên

1. Giới thiệu cấu trúc chương trình HelloWorld
2. Giới thiệu về file Program.cs
3. Giới thiệu về namespace, class và method
4. Câu lệnh Console.Writeline()

Tham khảo: <https://docs.microsoft.com/en-us/dotnet/core/tutorials/top-level-templates>

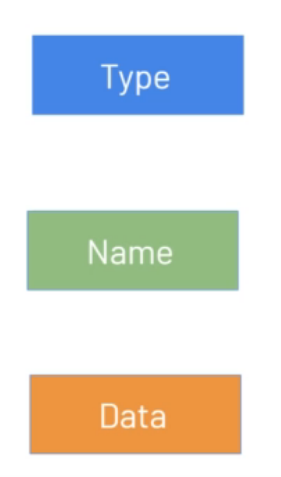
# Bài 4: Tổng quan về khái niệm kiểu dữ liệu và biến

## Khái niệm về biến

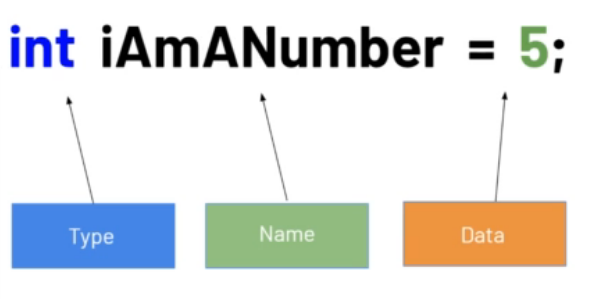
* Biến là một khái niệm cơ bản và quan trọng nhất của tất cả các ngôn ngữ lập trình.
* Biến là một không gian chứa một giá trị dữ liệu, và có thể được gán đi gán lại các giá trị khác nhau trên cùng 1 biến.
* Biến cấu tạo bởi kiểu dữ liệu, tên biến và dữ liệu lưu trong biến (có thể có hoặc chưa có)

## Khái niệm kiểu dữ liệu

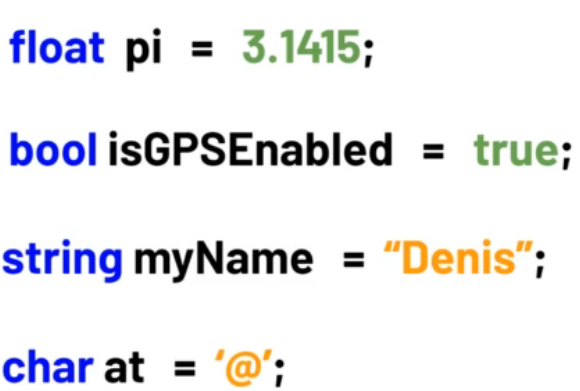


* Các bạn quan sát ở trên bàn có rất nhiều các loại thức ăn khác nhau như thị, rau, nước chấm, nước hoa quả…đây chính là các loại kiểu liệu khác nhau vì chúng có các đặc tính vật lý khác nhau.
* Chúng ta cần có các vật chứa khác nhau để chứa các loại thức ăn khác nhau: như cốc để đựng nước hoa quả, đĩa để đựng thịt, nồi để đựng nước lẩu, bát để đựng nước chấm 🡪 Đây chính là các biến để lưu trữ dữ liệu.
* Như vậy với mỗi loại dữ liệu chúng ta cần một kiểu dữ liệu tương ứng để lưu trữ loại dữ liệu đó. Ví dụ: Không thể để nước hoa quả vào một cái đĩa hay thịt vào một cái cốc nước. Vẫn được nhưng không ai làm thế cả.
* 

## Cách khai báo biến



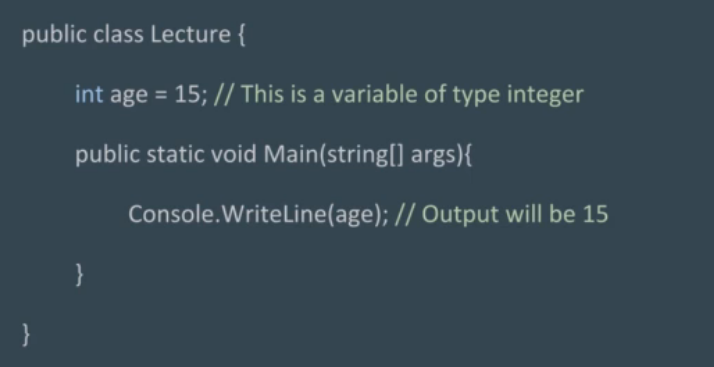
## Cách khai báo biến với các kiểu dữ liệu tiêu biểu



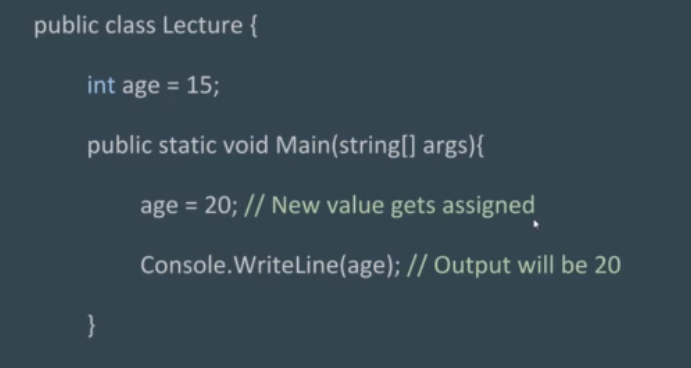
# Bài 5: Tìm hiểu về các kiểu dữ liệu hay dùng nhất.

## Khai báo giá trị với giá trị mặc định

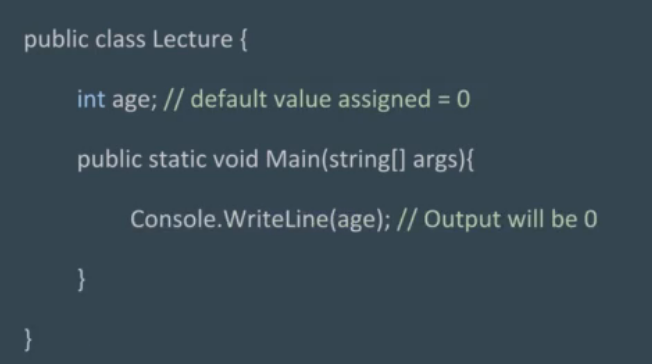
1. Gán giá trị mặc định cho biến khi khai báo



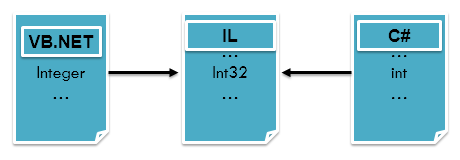
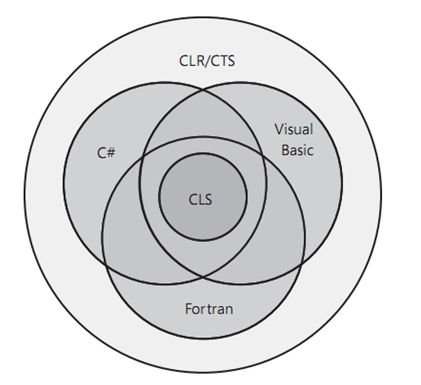
1. Gán lại giá trị cho biến



1. Khai báo mà không gán thì biến sẽ có giá trị là giá trị mặc định của kiểu dữ liệu đó.



## Khái niệm CLS và CTS

1. Common Type System (CTS): .NET framework hỗ trợ nhiều ngôn ngữ và đều dùng một thành phần gọi là hệ thống kiểu chung CTS trong CLR. CTS hỗ trợ một loạt kiểu và toán tử có thể thấy trong hầu hết các ngôn ngữ lập trình nên gọi một ngôn ngữ từ một ngôn ngữ khác sẽ không yêu cầu chuyển kiểu. Dẫn đến chúng ta có thể xây dựng các ứng dụng .NET sử dụng cả ngôn ngữ VB.NET lẫn C#, C++…
2. 
3. Common Language Specification (CLS): Đặc tả ngôn ngữ chung CLS là một tập con của CTS, nó định nghĩa một tập các quy tắc cho phép liên kết hoạt động trên nền tảng .NET. Các quy tắc này sẽ trợ giúp và chỉ dẫn cho các nhà thiết kế compiler của hãng thứ 3 hoặc những người muốn xây dựng thư viện dùng chung.
4. 

## Bảng tổng hợp một số kiểu dữ liệu cơ bản

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nhóm** | **Kiểu dữ liệu** | **Kích thước**  **(bytes)** | **Ý nghĩa** | **.NET Type (CTS)** |
| Kiểu số nguyên  (Giá trị mặc định là 0) | byte | 1 | Số nguyên dương**không dấu** có giá trị từ **0** đến **255.** | [System.Byte](https://docs.microsoft.com/en-us/dotnet/api/system.byte) |
| sbyte | 1 | Số nguyên có dấu có giá trị từ**-128**đến **127** | [System.SByte](https://docs.microsoft.com/en-us/dotnet/api/system.sbyte) |
| short | 2 | Số nguyên có dấu có giá trị từ**-32,768**đến **32,767** | [System.Int16](https://docs.microsoft.com/en-us/dotnet/api/system.int16) |
| ushort | 2 | Số nguyên **không dấu**có giá trị từ **0** đến **65,535** | [System.UInt16](https://docs.microsoft.com/en-us/dotnet/api/system.uint16) |
| int | 4 | Số nguyên có dấu có giá trị từ **-2,147,483,647** đến **2,147,483,647** | [System.Int32](https://docs.microsoft.com/en-us/dotnet/api/system.int32) |
| uint | 4 | Số nguyên**không dấu**có giá trị từ **0** đến **4,294,967,295** | [System.UInt32](https://docs.microsoft.com/en-us/dotnet/api/system.uint32) |
| long | 8 | Số nguyên có dấu có giá trị từ **-9,223,370,036,854,775,808** đến **9,223,370,036,854,775,807** | [System.Int64](https://docs.microsoft.com/en-us/dotnet/api/system.int64) |
| ulong | 8 | Số nguyên **không dấu**có giá trị từ **0**đến **18,446,744,073,709,551,615** | [System.UInt64](https://docs.microsoft.com/en-us/dotnet/api/system.uint64) |
| Kiểu ký tự  (Giá trị mặc định là '\0') | char | 2 | Chứa một ký tự Unicode | [System.Char](https://docs.microsoft.com/en-us/dotnet/api/system.char) |
| Kiểu logic  (Giá trị mặc định là false) | bool | 1 | Chứa 1 trong 2 giá trị logic là **true**hoặc **false** | [System.Boolean](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) |
| Kiểu số thực  (Giá trị mặc định là 0) | float | 4 | Kiểu số thực dấu chấm động có giá trị dao động  từ **±1.5 x 10−45 to ±3.4 x 1038**, với 7 chữ số có nghĩa.  Thường sử dụng cho các thư viện đồ hoạ cần yêu cầu sức mạnh xử lý cao. | [System.Single](https://docs.microsoft.com/en-us/dotnet/api/system.single) |
| double | 8 | Kiểu số thực dấu chấm động có giá trị dao động từ**±5.0 × 10−324 to ±1.7 × 10308**, với 15, 16 chữ số có nghĩa. Thường dùng cho các tính toán thực tế phổ biến ngoại trừ tài chính. | [System.Double](https://docs.microsoft.com/en-us/dotnet/api/system.double) |
| decimal | 16 | Kiểu số thực có dấu chấm động có giá trị giao động từ **±1.0 x 10-28 to ±7.9228 x 1028**. Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính. Thường dùng cho các ứng dụng tài chính có độ chính xác cao. | [System.Decimal](https://docs.microsoft.com/en-us/dotnet/api/system.decimal) |

## Kiểu string

* Kiểu string khác với các kiểu trên là kiểu dữ liệu tham chiếu dùng để lưu chuỗi ký tự văn bản.
* Nếu không gán giá trị thì mặc định giá trị của kiểu String sẽ là null

## Lưu ý

* Kiểu dữ liệu có miền giá trị lớn hơn sẽ chứa được kiểu dữ liệu có miền giá trị nhỏ hơn. Như vậy biến kiểu dữ liệu nhỏ hơn có thể gán giá trị qua biến kiểu dữ liệu lớn hơn (sẽ được trình bày trong phần tiếp theo).
* Giá trị của kiểu char sẽ nằm trong dấu **‘ ’** (nháy đơn).
* Giá trị của kiểu string sẽ nằm trong dấu **“ ”**(nháy kép).
* Giá trị của biến kiểu float phải có chữ **F** hoặc **f**làm hậu tố.
* Giá trị của biến kiểu decimal phải có chữ **m**hoặc **M**làm hậu tố.
* Trừ kiểu string, tất cả kiểu dữ liệu trên đều không được có giá trị null:
  + Null là giá trị rỗng, không tham chiếu đến vùng nhớ nào.
  + Để có thể gán giá trị null cho biến thì ta thêm ký tự ? vào sau tên kiểu dữ liệu là được. Ví dụ: int? hay bool? . . .

# Bài 6: Thực hành về kiểu int, float, double

int number1 = 10;

int number2 = 12;

int number3, number4, number5;

int sum = number1 + number2;

Console.WriteLine("Sum of " + number1 + " and " + number2 + ": " + sum);

float floatNumber1 = 1.2f;

float floatNumber2 = 1.3f;

float sumFloat = floatNumber1 + floatNumber2;

float divFloat = floatNumber1/ floatNumber2;

float multiFloat = floatNumber1 \* floatNumber2;

Console.WriteLine("Sum of " + floatNumber1 + " and " + floatNumber2 + ": " + sumFloat);

Console.WriteLine("Division of " + floatNumber1 + " for " + floatNumber2 + ": " + divFloat);

Console.WriteLine("Multiple of " + floatNumber1 + " for " + floatNumber2 + ": " + multiFloat);

Console.Read();

# Bài 7: Kiểu dữ liệu String

string myName = "Bach Ngoc Toan";

string[] myNames = myName.Split(' ');

Console.WriteLine("My name is: " + myName);

Console.Read();

# Bài 8: Quy tắc code chuẩn

1. Quy tắc đặt tên
   1. Tên file
   2. Tên biến
   3. Tên phương thức
   4. Tên class
2. Comment
   1. Comment 1 dòng
   2. Comment nhiều dòng
   3. Comment XML

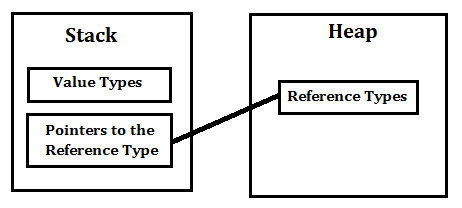
Tham khảo: <https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

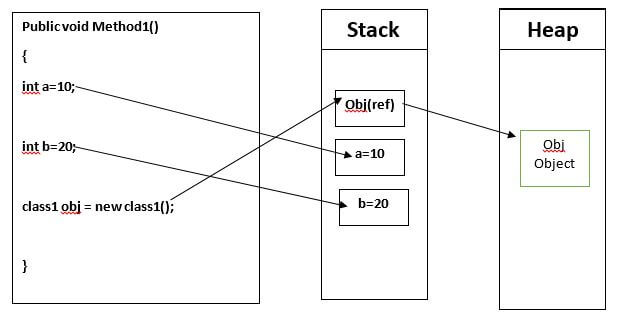
<https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md>

# Bài 9: Kiểu tham trị và kiểu tham chiếu

Trong các kiểu của .NET Framework chúng ta có 2 kiểu là Value Type (kiểu tham trị) và Reference Type (kiểu tham chiếu).

* Kiểu tham trị lưu trực tiếp dữ liệu trong bộ nhớ Stack
* Kiểu tham chiếu chỉ lưu địa chỉ trong Stack còn giá trị biến nằm ở nơi khác (Heap)





### Kiểu tham trị (Value Type)

Một kiểu tham trị lưu nội dung của nó trong bộ nhớ cấp phát là Stack. Khi chúng ta tạo một biến kiểu tham trị thì một vùng nhớ sẽ được cấp phát để lưu giá trị của biến một cách trực tiếp.

Nếu bạn gán nó cho một biến khác thì giá trị sẽ được copy trực tiếp và cả 2 biến sẽ làm việc độc lập.

Các kiểu dữ liệu tham trị trong .NET:

* Các kiểu số nguyên
* Kiểu số thực
* Kiểu logic bool
* Kiểu ký tự char
* Kiểu struct
* Enum

### Kiểu tham chiếu

Kiểu tham chiếu được dùng để lưu một giá trị tham chiếu (địa chỉ ô nhớ) của một đối tượng nhưng không lưu trữ đối tượng đó.

Bởi vì kiểu tham chiếu chỉ lưu địa chỉ của ô nhớ của biến thay vì lưu giá trị của biến, nên khi gán một biến tham chiếu cho một biến khác thì nó không copy data mà nó chỉ copy địa chỉ tham chiếu.

Nên cả 2 biến sẽ cùng tham chiếu đến một địa chỉ giống nhau trên bộ nhớ Heap.

Điều này có nghĩa khi một biến tham chiếu không được dùng nữa, nó sẽ được đánh dấu cho Garbage collection.

Các kiểu dữu liệu tham chiếu:

* Classs
* Object
* Array
* Indexer
* Interface

### Bộ nhớ Stack và Heap

Stack là bộ nhớ cấp phát tĩnh còn Heap là bộ nhớ cấp phát động, cả 2 đều được lưu trữ trên RAM của máy tính.

Bạn có thể dùng Stack nếu bạn biết chắc độ lớn của dữ liệu cần lưu trước khi biên dịch chương trình, nó không được quá lớn.

Bạn có thể dùng heap nếu bạn không biết chính xác độ lớn dữ liệu bạn sẽ cần ở lúc chạy hoặc nếu bạn cần cấp phát rất nhiều dữ liệu.

Trong ứng dụng mutilple thread, mỗi một thread sẽ có một stack riêng nhưng chúng sẽ chia sẻ bộ nhớ heap. Stack là riêng cho từng thread còn heap là chia sẻ cho toàn ứng dụng.

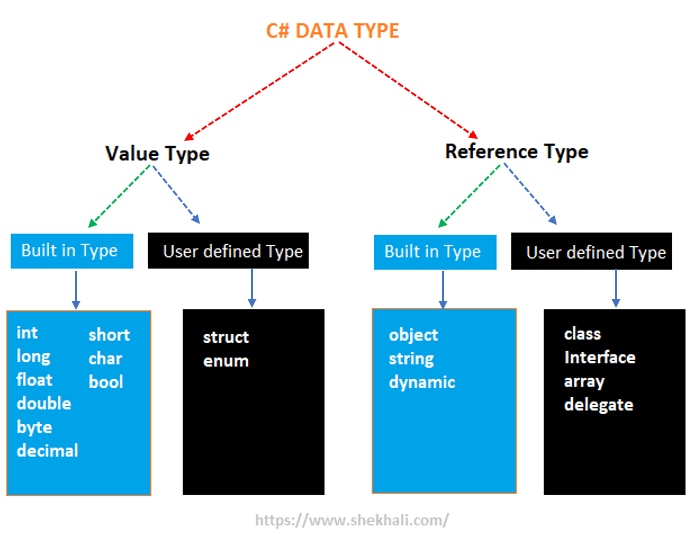
### So sánh giữa Stack và Heap

|  |  |
| --- | --- |
| Stack | Heap |
| Bộ nhớ được quản lý tự động | Bộ nhớ được quản lý bằng tay |
| Kích cỡ nhỏ | Kích cỡ lớn |
| Truy cập dễ dàng và nhanh chóng, dễ dàng cache | Khó cache vì bị phân tán trong bộ nhớ |
| Không linh hoạt, bộ nhớ cấp phát không thể thay đổi | Lich hoạt, bộ nhớ cấp phát có thể đổi |
| Giới hạn trong phạm vi thread | Được truy cập toàn bộ ứng dụng |
| Hệ điều hành cấp phát stack khi thread được tạo | Hệ điều hành được gọi bởi ngôn ngữ lúc chạy ứng dụng để cấp phát heap cho ứng dụng. |

### So sánh cấp phát tĩnh và cấp phát động

|  |  |
| --- | --- |
| **Cấp phát tĩnh** | **Cấp phát động** |
| Kích thước phải biết lúc biên dịch | Không biết kích thước biến lúc biên dịch |
| Thực hiện lúc biên dịch | Thực hiện lúc runtime |
| Được gán cho stack | Gán cho heap |
| FILO (First – in, last- out) | Không có thứ tự |

### Tóm lại



Chúng ta có 2 kiểu dữ liệu là tham trị và tham chiếu:

1. Tham trị là các kiểu dữ liệu kích thước nhỏ, size cố định giá trị lưu trực tiếp trên bộ nhớ Stack.
2. Tham chiếu là kiểu chỉ lưu địa chỉ trên stack và giá trị lưu trên heap.
3. Cả 2 loại bộ nhớ này đều được lưu trên RAM, nó chỉ là khác vùng thôi. Một bên là cấp phát tĩnh 1 bên là cấp phát động.

#### Tại sao lại phải chia ra thế?

Vì với kiểu dữ liệu lớn, như là object thì không thể biết kích thước của nó lúc biên dịch nên không lưu trên stack mà chỉ lưu địa chỉ của Heap thôi, vì heap là bộ nhớ cấp phát động.

Ví dụ: Ta có đọc 1 cuốn sách Harry Porter, thay vì tôi nói tôi đã đọc cuốn sách **Harry Porter 1** thì tôi phải nói là **Tôi đọc …. từ câu đầu cho đến câu cuối…**bạn có biết không? Thay vì chúng ta phải nêu ra toàn bộ nội dung cuốn sách rất dài thì ta chỉ cần refer đến tên cuốn sách, chính là địa chỉ ô nhớ trỏ đến nội dung cuốn sách. Còn cuốn sách có nội dung như thế nào thì bạn đó sẽ tự tìm đọc.

Còn những câu văn ngắn thì chúng ta có thể nói luôn trong cuộc trò chuyện. Ví dụ: Bạn ăn cơm chưa? Thì câu đó không nằm trong cuốn sách nào hoặc nó rất ngắn nên có thể nói luôn trong cuộc hội thoại.

Đó là kiểu tham trị.

# Bài 10: Nhập xuất cơ bản với màn hình Console

### Ứng dụng Console là gì?

Trích dẫn nguyên văn: “A console application, in the context of C#, is an application that takes input and displays output at a command line console with access to three basic data streams: standard input, standard output and standard error. A console application facilitates the reading and writing of characters from a console - either individually or as an entire line. It is the simplest form of a C# program and is typically invoked from the Windows command prompt. A console application usually exists in the form of a stand-alone executable file with minimal or no graphical user interface (GUI).”

Nguồn: <https://www.techopedia.com/definition/25593/console-application-c>

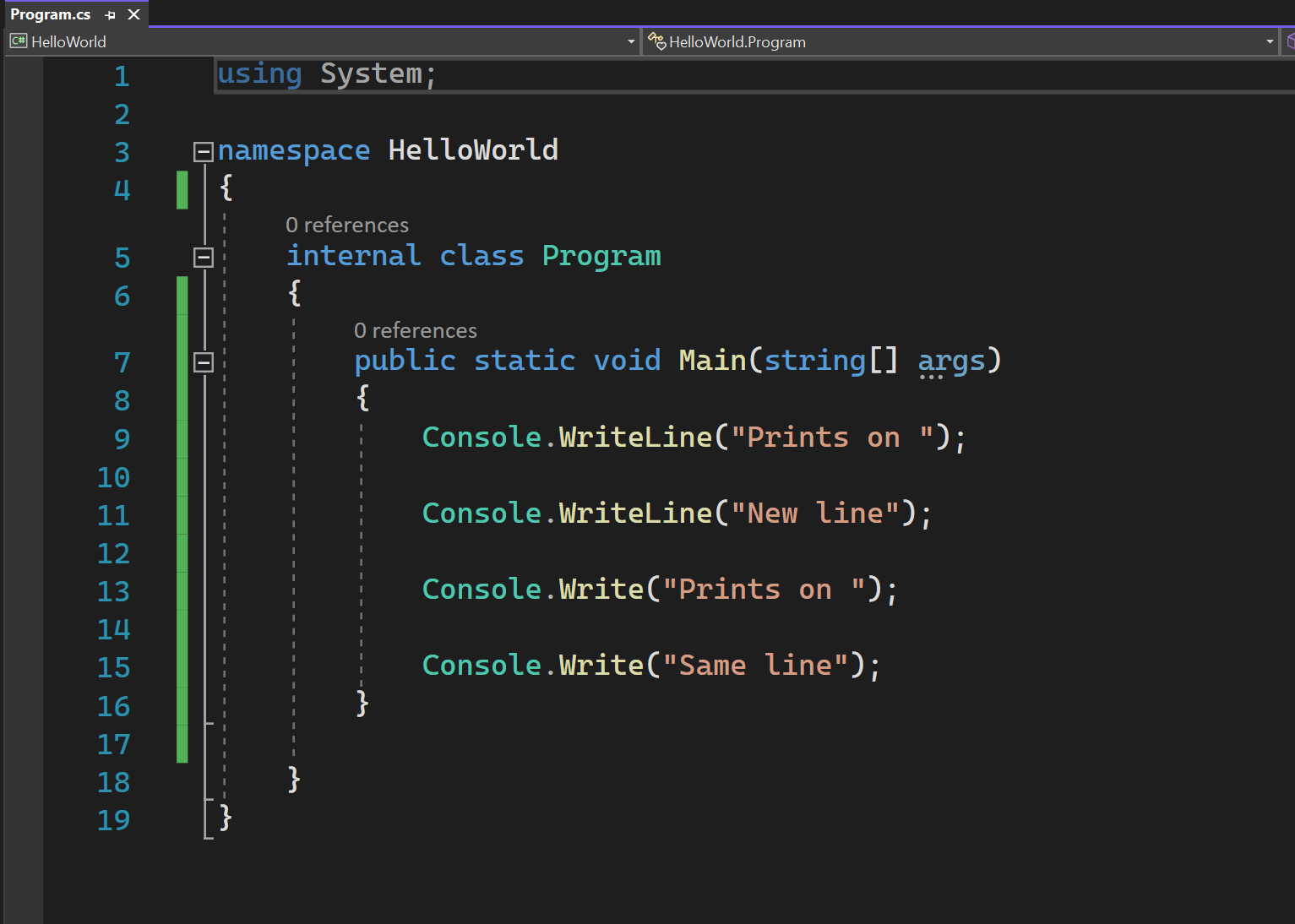
Để tạo ứng dụng Console, chúng ta cần làm việc với Class System.Console. Trong đó, System là một namespace, Console là lớp bên trong namespace System.

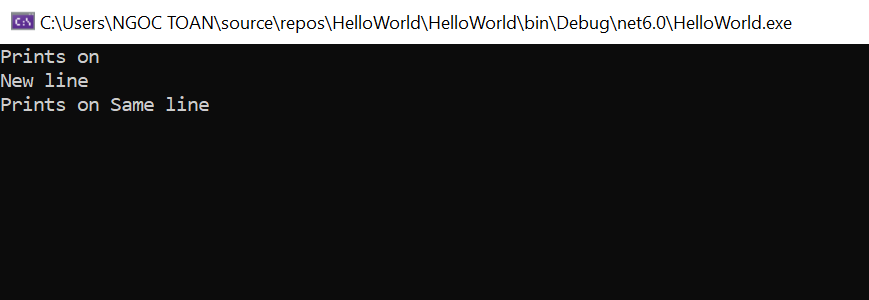
### Xuất dữ liệu với Console

Để xuất nội dung trong C#, chúng ta có thể sử dụng:

* System.Console.WriteLine()
* System.Console.Write()

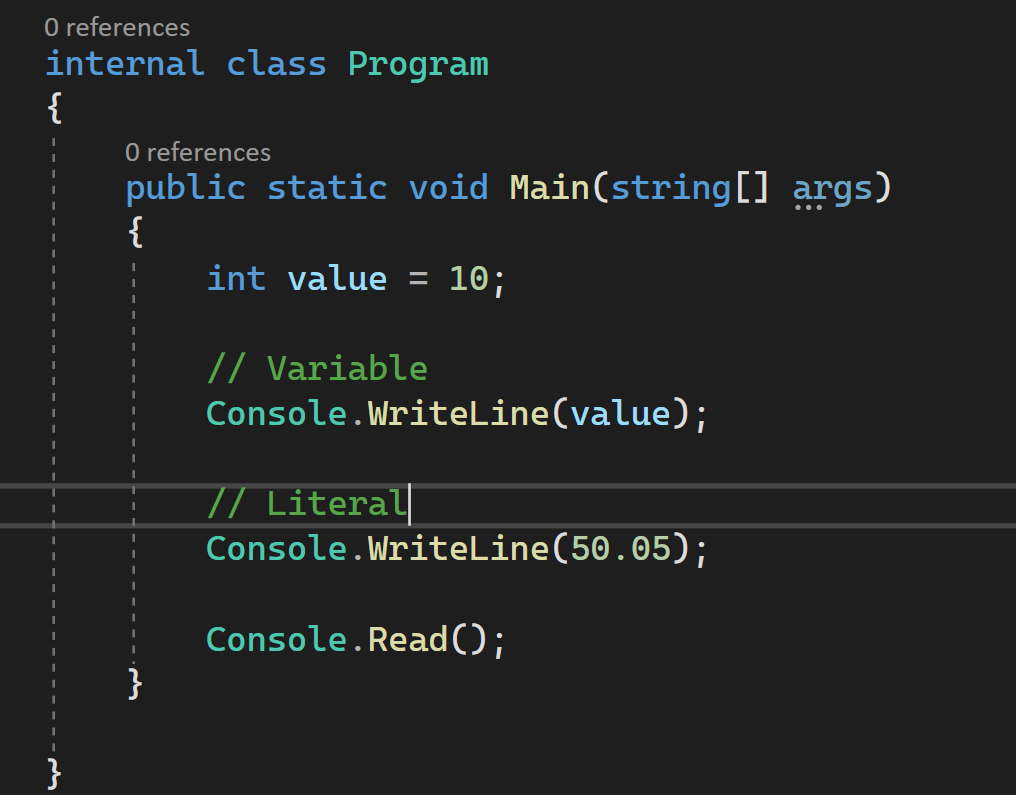
Sự khác biệt cơ bản của WriteLine() và Write() là phương thức Write() chỉ in chuỗi được cung cấp, trong khi phương thức WriteLine() in chuỗi và chuyển đến đầu dòng tiếp theo.

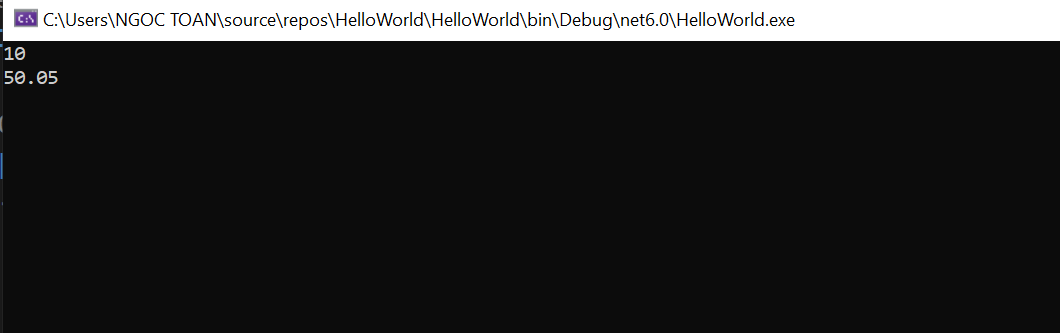




#### In Variable và Literal sử dụng phương thức WriteLine() và Write()

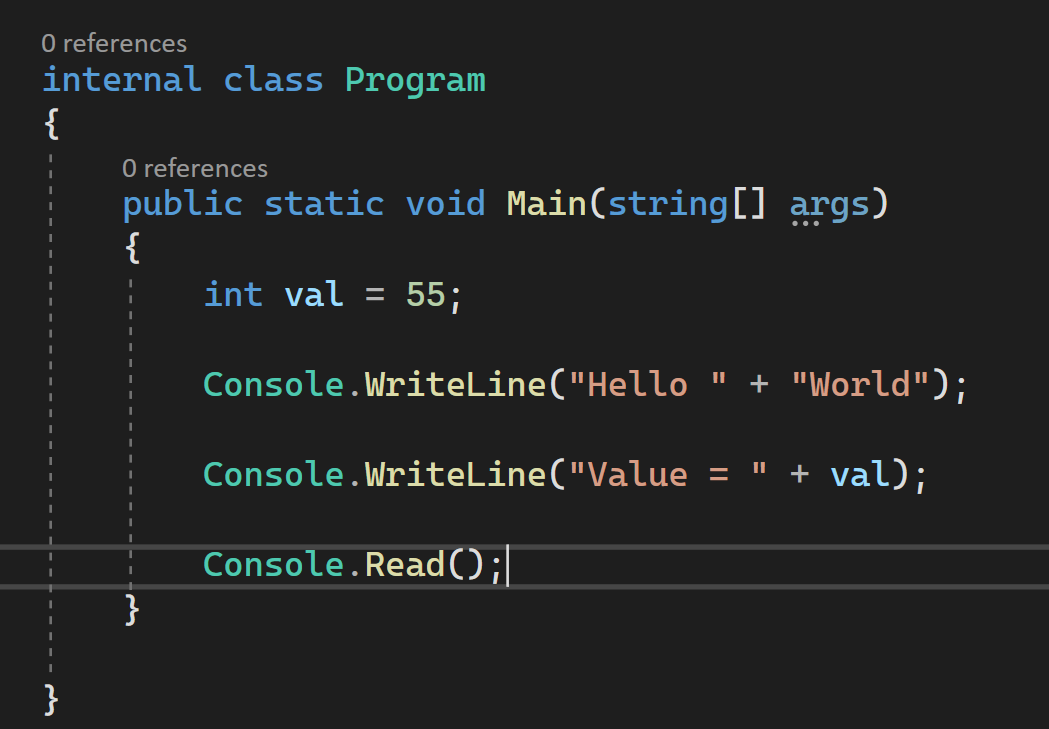
Phương thức WriteLine() và phương thức Write() có thể sử dụng để in biến và hằng. Hãy xem ví dụ dưới đây:

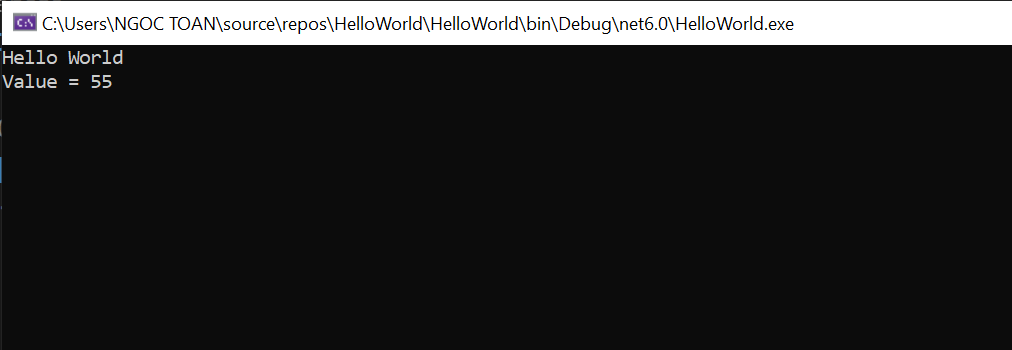




#### Kết hợp (nối) hai chuỗi sử dụng toán tử cộng (+) và in chúng

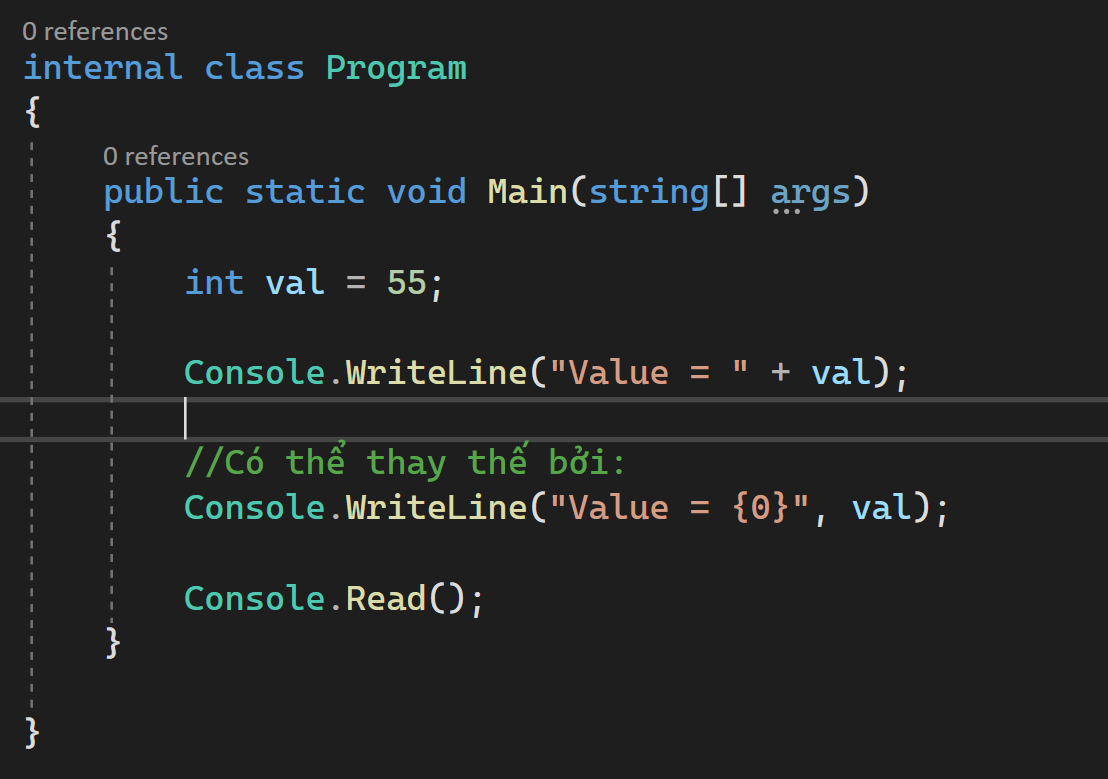
Các chuỗi có thể được kết hợp / nối (concatenated string) bằng cách sử dụng toán tử + trong khi in.





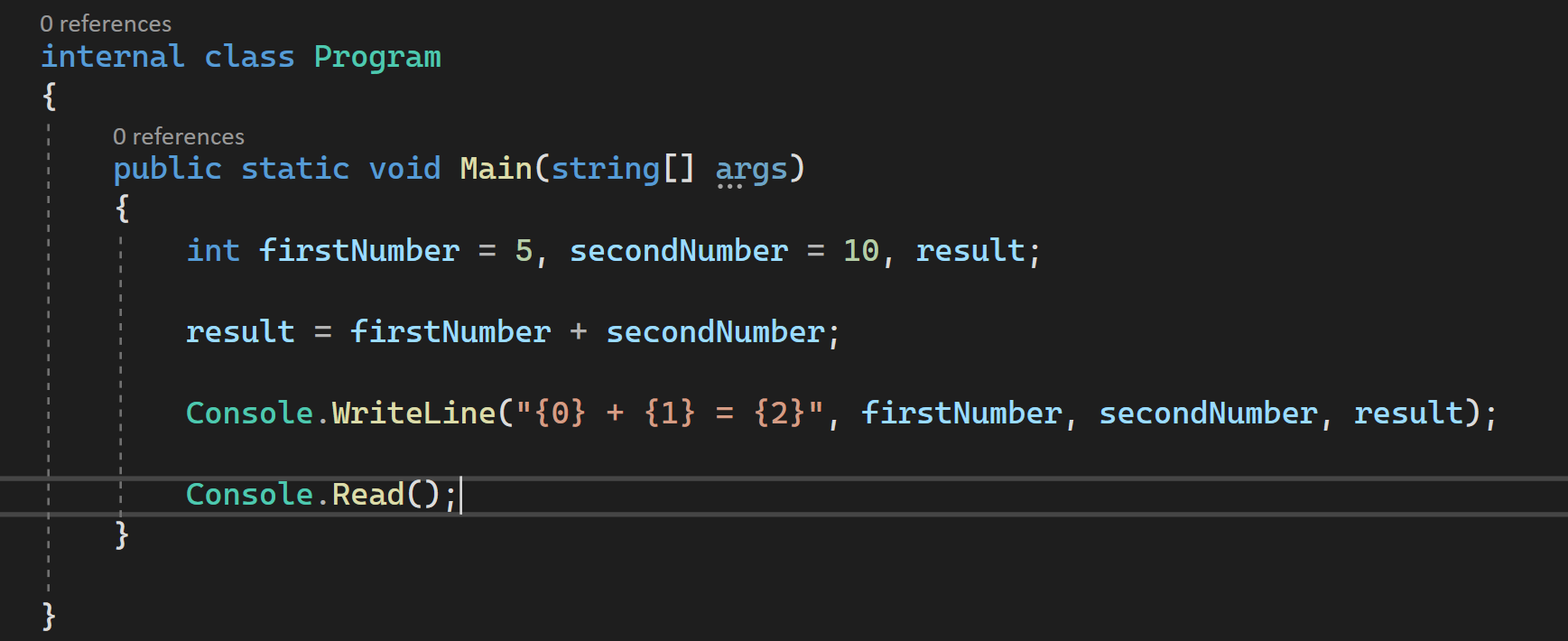
#### In chuỗi kết hợp bằng chuỗi đã được định dạng (Formatted String)

Một giải pháp tốt hơn để in chuỗi kết hợp là sử dụng chuỗi đã được định dạng. Formatted string cho phép lập trình viên sử dụng trình giữ chỗ (placeholder) cho các biến.



**{0}** là trình giữ chỗ cho biến val , nó sẽ bị thay thế bởi giá trị của val. Chỉ có 1 biến được sử dụng nên chỉ có một placeholder.

Ví dụ dưới đây là nhiều biến:



Khi chạy chương trình, output sẽ hiển thị

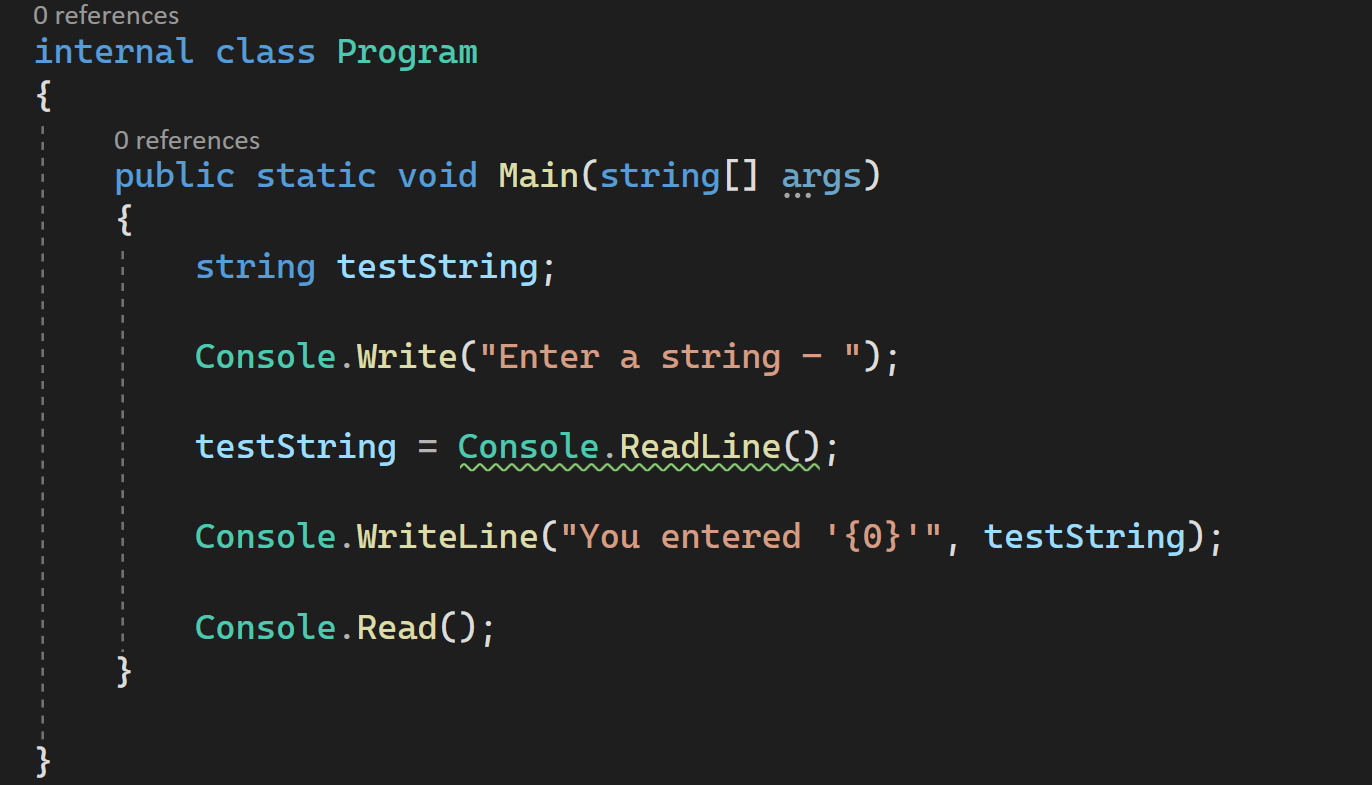
5 + 10 = 15

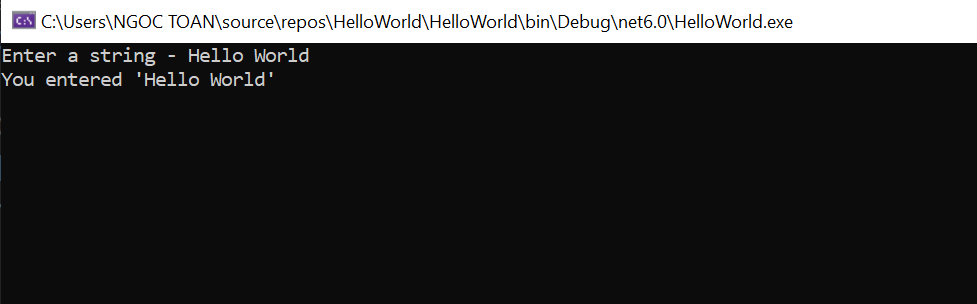
Ở đây, {0} bị thay thế bởi firstNumber, {1} bịthay thế bởi secondNumber và {2} bị thay thế bởi result.

Cách tiếp cận output in này dễ đọc hơn và ít bị lỗi hơn toán tử +.

### Nhập dữ liệu với Console

Trong C#, cách đơn giản nhất để lấy input từ user là sử dụng phương thức ReadLine() của lớp Console. Tuy nhiên, Read() và ReadKey() cũng có thể làm điều đó được. Hai phương thức này cùng thuộc lớp Console.

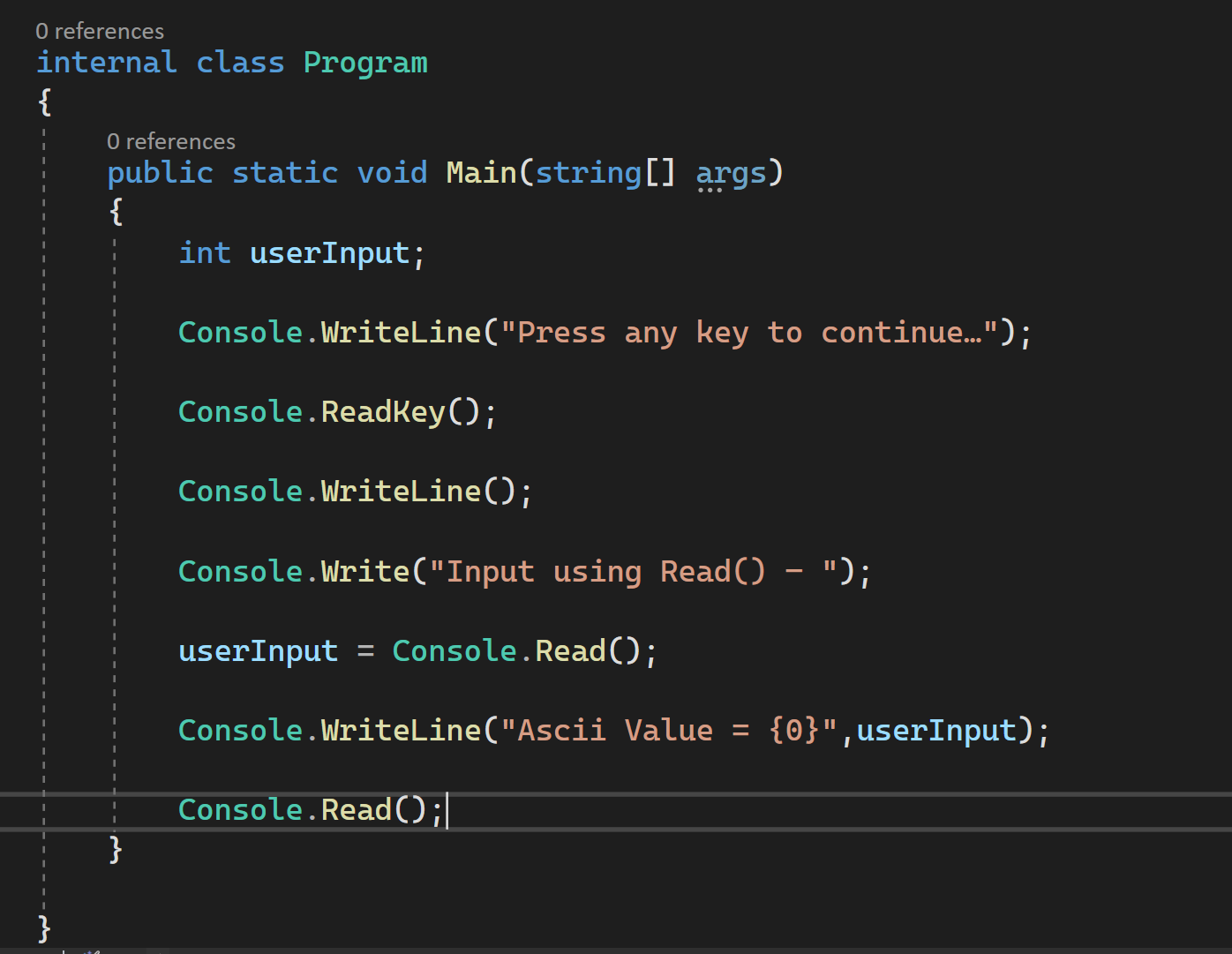


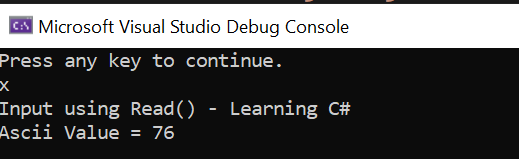


#### Sự khác nhau của phương thức ReadLine(), Read() và ReadKey()

* **ReadLine():** Phương thức ReadLine() đọc dòng tiếp theo của input (từ dòng input chuẩn). Nó trả về cùng một chuỗi.
* **Read():** Phương thức Read() đọc ký tự tiếp theo từ dòng input chuẩn. Nó trả về giá trị Ascii của ký tự.
* **ReadKey():** Phương thức ReadKey() tiếp nhận phím tiếp theo mà user nhấn. Phương pháp này thường được sử dụng để giữ màn hình không hiển thị output cho đến khi người dùng nhấn một phím.

Ví dụ: Sự khác nhau của phương thức Read() và ReadKey():





Từ ví dụ trên, bạn có thể thấy cách hoạt động của Console.ReadKey() và Console.Read(). Khi sử dụng Console.ReadKey(), khi nhấn phím bất kỳ màn hình sẽ hiển thị kết quả.

Khi sử dụng Console.Read(), bạn sẽ phải viết cả dòng lệnh trong khi nó chỉ trả về giá trị ASCII value của ký tự đầu tiên. Trong ví dụ này, 76 là giá trị ASCII của L.

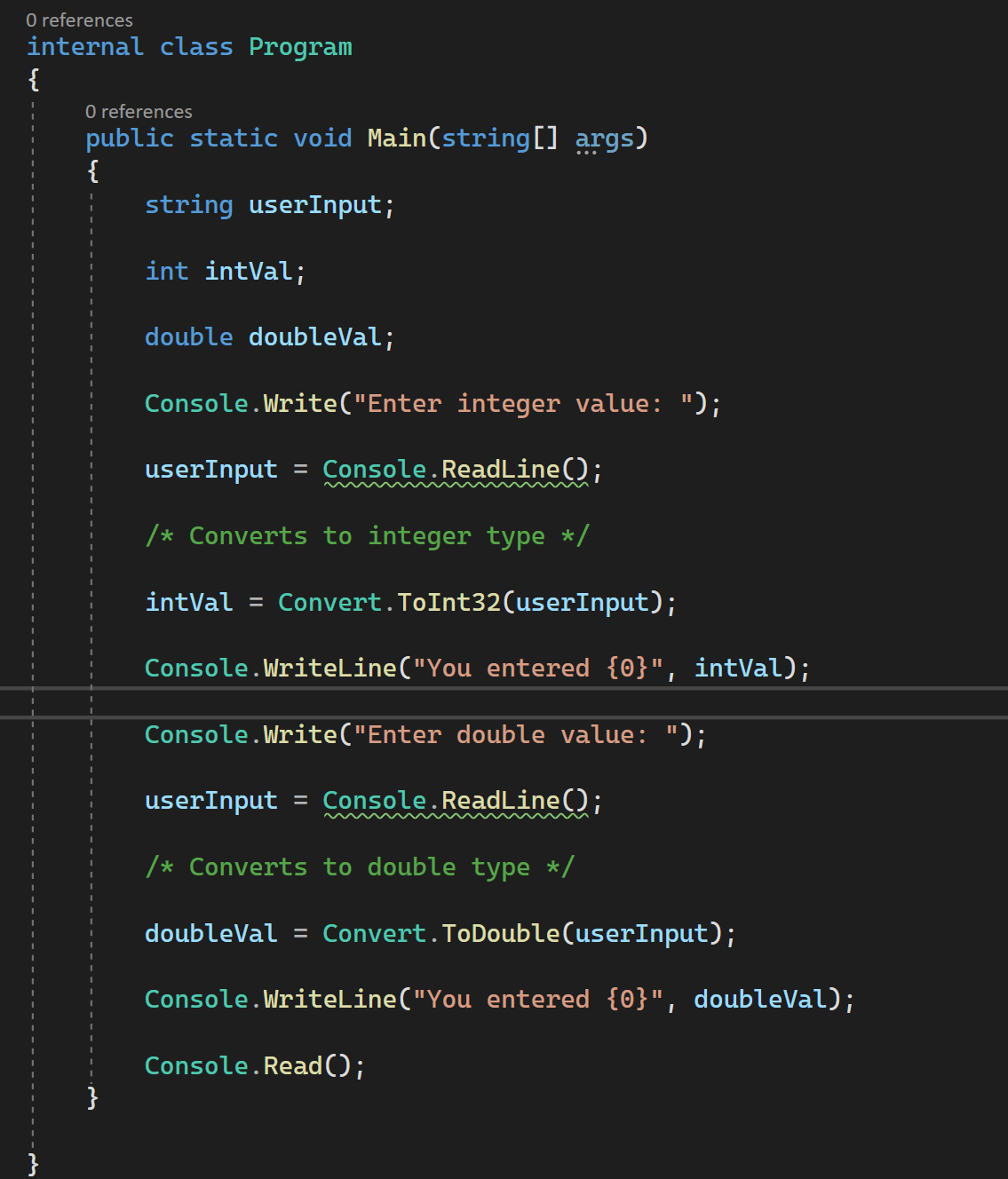
#### Đọc giá trị số

Việc đọc một ký tự hoặc chuỗi rất đơn giản trong C #. Tất cả những gì bạn cần làm là gọi đúng tên các phương thức tương ứng.

Tuy nhiên, việc đọc các giá trị số có thể khá phức tạp.

Chúng ta sẽ vẫn sử dụng cùng một phương thức ReadLine(). Nhưng vì phương thức này nhận đầu vào là chuỗi, nên nó cần được chuyển đổi thành kiểu số nguyên hoặc dấu chấm động.

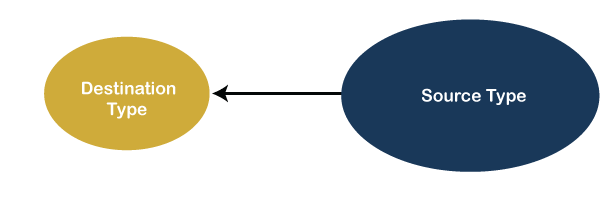
Một cách tiếp cận đơn giản để chuyển đổi đầu vào là sử dụng các phương thức của lớp Convert .



Phương thức **ToInt32()** và **ToDouble()** của Convert class chuyển đổi input thành kiểu integer và double. Tương tự, chúng ta có thể chuyển đổi input sang các kiểu khác.

# Bài 11: Chuyển đổi kiểu dữ liệu (Type Casting)

Chuyển đổi kiểu dữ liệu là khi bạn muốn gán giá trị của một kiểu dữ liệu này sang kiểu dữ liệu khác



Trong C# có 2 cách chuyển đổi kiểu dữ liệu:

## Implicit Casting (chuyển đổi ngầm định tự động)

Chuyển đổi kiểu dữ liệu range nhỏ hơn sang kiểu dữ liệu range lớn hơn.

char -> int -> long -> float -> double

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

Console.WriteLine(myInt); // Outputs 9

Console.WriteLine(myDouble); // Outputs 9

## Explicit Casting (chuyển đổi tường minh bằng tay)

Chuyển đổi từ kiểu dữ liệu lớn hơn sang kiểu dữ liệu nhỏ hơn

double -> float -> long -> int -> char

int five = 5;

var doubleFive = (double)five;

char a = 'a';

var valueA = (int)a;

float myFloat = 4.56F;

decimal myMoney = (decimal)myFloat;

## Khác nhau giữa casting, parsing và converting

### Casting: là chuyển 1 giá trị từ kiểu này sang kiểu khác hoặc xuất ra lỗi

int five = 5;

var doubleFive = (double)five;

char a = 'a';

var valueA = (int)a;

float myFloat = 4.56F;

decimal myMoney = (decimal)myFloat;

### Conversion: Là cố chuyển một kiểu đối tượng sang kiểu khác, ít lỗi hơn nhưng chậm hơn

int five = 5;

decimal decFive = Convert.ToDecimal(five);

decimal myMoney = 5.67M;

int intMoney = Convert.ToInt32(myMoney); //Value is now 6;

//the decimal value was rounded

### Parsing: Là cố chuyển một chuỗi sang một kiểu nguyên thuỷ

string testString = "10.22.2000";

double decValue = double.Parse(testString); //Exception thrown here!

string intTest = "This is a test string";

int intValue = int.Parse(intTest); //Exception thrown here!

string value = "5.0";

decimal result;

bool isValid = decimal.TryParse(value, out result);

### Khác nhau giữa Parse và Convert

string s1 = "1234";

string s2 = "1234.65";

string s3 = null;

string s4 = "123456789123456789123456789123456789123456789";

result = Int32.Parse(s1); //1234

result = Int32.Parse(s2); //FormatException

result = Int32.Parse(s3); //ArgumentNullException

result = Int32.Parse(s4); //OverflowException

result = Convert.ToInt32(s1); // 1234

result = Convert.ToInt32(s2); // FormatException

result = Convert.ToInt32(s3); // 0

result = Convert.ToInt32(s4); // OverflowException

## TryParse()

Trong trường hợp chúng ta không biết liệu một string có thể chuyển sang kiểu nào đó hay không thì chúng ta có thể dùng TryParse()

string value = "5.0";

decimal result;

bool isValid = decimal.TryParse(value, out result);

Nếu giá trị biến isValid là true có nghĩa là chuỗi có thể chuyển thành công, còn ngược lại thì là fail. Từ khoá out chúng ta sẽ học trong phần sau.

## Từ khoá mới

### Từ khoá is: Sử dụng để kiểm tra nếu một giá trị cụ thể là một kiểu cụ thể

var myValue = 6.5M; //M literal means type will be decimal

if (myValue is decimal) { /\*...\*/ }

### Từ khoá as: Sử dụng để chuyển một object từ một kiểu sang một kiểu khác

Ví dụ 1:

string testString = "This is a test"; //string is a reference type

object objString = (object)testString; //Cast the string to an object

string test2 = objString as string; //Will convert to string successfully

Ví dụ 2:

public class ClassA { /\*...\*/ }

public class ClassB { /\*...\*/ }

var myClass = new ClassA();

var newClass = myClass as ClassB; //Exception thrown here!

Ví dụ 3:

public class ClassA { /\*...\*/ }

public class ClassB : ClassA { /\*...\*/ }

var myClass = new ClassB();

var convertedClass = myClass as ClassA;

### Từ khoá typeof: Trả về kiểu của một đối tượng

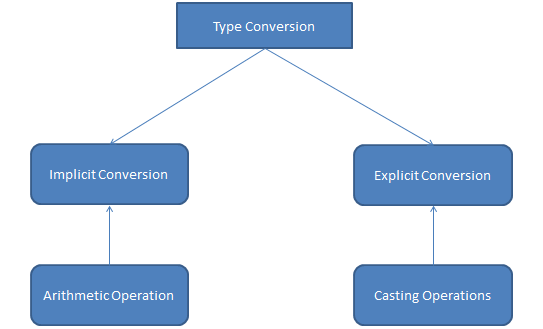
var sentence = "This is a sentence.";

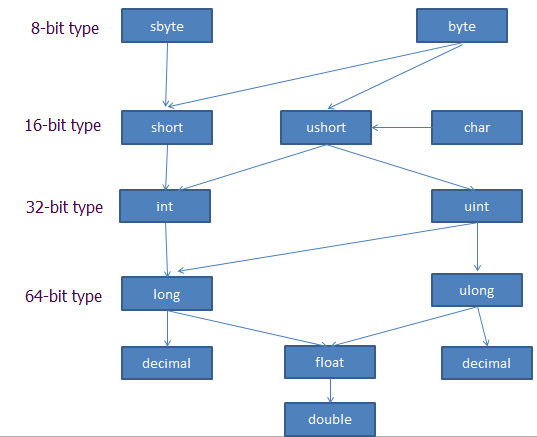
var type = sentence.GetType();

if (type == typeof(string)) { /\*...\*/ }

else if (type == typeof(int)) { /\*...\*/ }

## Tổng kết





# Bài 12: Hằng số (const)

Tham khảo tài liệu: <https://docs.microsoft.com/vi-vn/dotnet/csharp/language-reference/keywords/const>

1. Bạn sử dụng từ khoá const để khai báo một trường là hằng số hoặc một local constant là hằng số.
2. Trường constant (constant field) và local constant không phải là các biến, chúng không thể bị thay đổi.
3. Constant có thể là số, giá trị boolean, string hay một tham chiếu null.
4. Không tạo một constant cho việc hiển thị thông tin mà bạn có thể thay đổi bất cứ khi nào.

using System;

namespace HelloWorld

{

internal class Program

{

// Constants as a fields

const double PI = 3.14159;

const int NumberOfWeeksInYear = 52;

const int NumberOfMonthsInYear = 12;

const string MyBirthDay = "2000-11-11";

public static void Main(string[] args)

{

double radius = 10;

Console.WriteLine("My birthday is {0}", MyBirthDay);

Console.WriteLine(radius \* radius \* PI);

Console.ReadKey();

}

}

}

# Bài 13: Bài tập tổng kết chương 1

Bài tập 1: Viết chương trình tính tổng 2 số nguyên được nhập vào từ người dùng sau đó in ra dòng chữ: “Sum of <a> and <b> is: <total>”. Trong đó giá trị của a, b và total là giá trị của biến.

Bài tập 2: Viết chương trình nhập vào họ tên, số điện thoại và giới tính để in ra thông tin người đó.

Bài tập 3: Viết chương trình tính diện tích hình tròn với bán kính được nhập vào từ bàn phím.

Review code email: tedu.international@gmail.com

*Dừng lại suy nghĩ, sau đó mới được xem lời giải.*

# Bài 14: Method

## Phương thức là gì?

Phương thức là một khối lệnh chứa một tập hợp các dòng lệnh. Tập hợp lệnh đó sẽ được thực thi thông qua lời gọi phương thức và chỉ ra tham số cho phương thức đó (nếu có).

Trong C# thì tất cả các lệnh được thực thi đều phải được nằm trong một phương thức.

Phương thức Main là điểm khởi đầu (entry point) chủa tất cả các các ứng dụng C# và nó được gọi bởi CLR khi chương trình bắt đầu chạy.

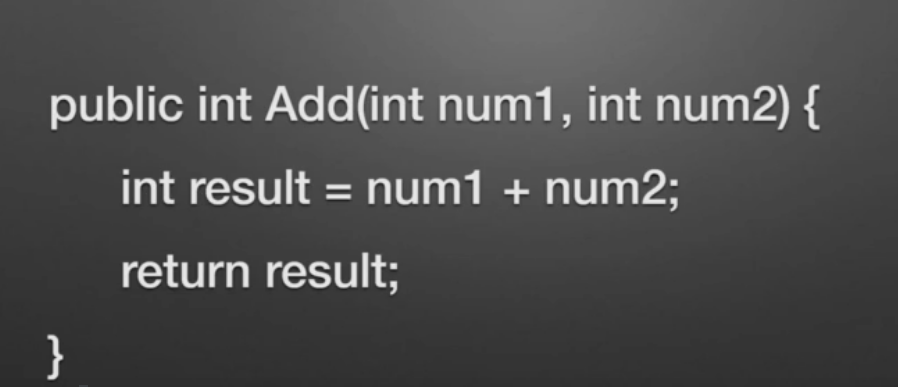
## Cú pháp khai báo

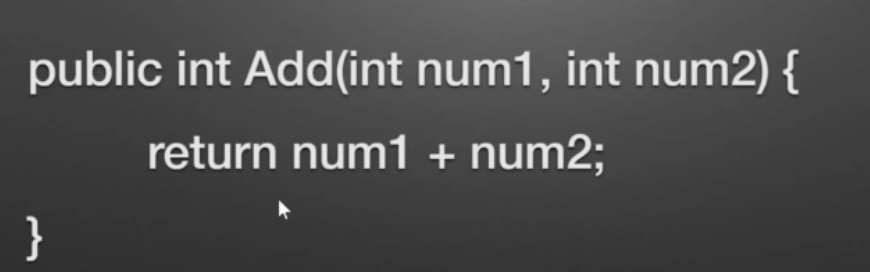


Trong đó:

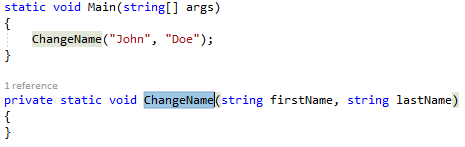
* Access specifier: quyết định mức độ hiện diện của biến hoặc phương thức với class khác.
* Return type: Một phương thức có thể có 1 giá trị trả về, kiểu dữ liệu trả về của phương thức. Nếu phương thức không trả về bất cứ giá trị nào thì nó sẽ là void.
* Method name: Là tên của phương thức, nó phải là duy nhất trong class và có phân biệt hoa thường. Nó không thể trùng với bất cứ khai báo nào trong class
* Parameter list: Bao bởi cặp ngoặc đơn, tham số được sử dụng để truyền và nhận dữ liệu từ một phương thức. Danh sách tham số bao gồm kiểu, thứ tự và số lượng tham số của phương thức. Tham số là tuỳ chọn vì có thể có phương thức không có bất cứ tham số nào.
* Method body: Là phần chứa tập lệnh cần thiết để xử lý nghiệp vụ trong phương thức.

Ví dụ:





## Void method



## Ví dụ thực hành

* Viết chương trình máy tính, cộng trừ nhân chia 2 số sử dụng phương thức với tham số và giá trị trả về.

# Bài 15: Exception Handler

## Khái niệm Exception

Xử lý ngoại lệ trong C# có nghĩa là xử lý các lỗi exceptions có thể xảy ra, giúp chương trình không bị gián đoạn.

Trong đó Exceptions là một sự kiện xảy ra khi một chương trình đang chạy (thực thi) , sự kiện đó làm cho luồng xử lý thông thường của chương trình không thể thực hiện một cách bình thường, thậm chí chết chương trình.

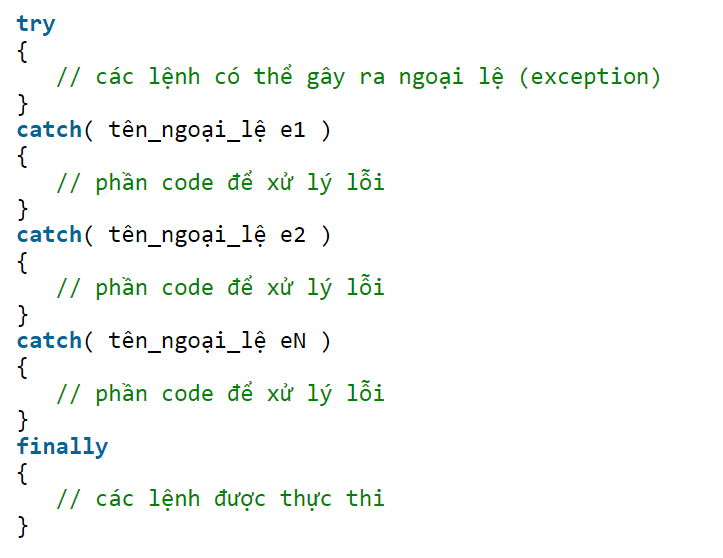
Một Exception trong C# là một phản hồi về một tình huống ngoại lệ mà xuất hiện trong khi một chương trình đang chạy, ví dụ như chia cho số 0.

## Từ khoá

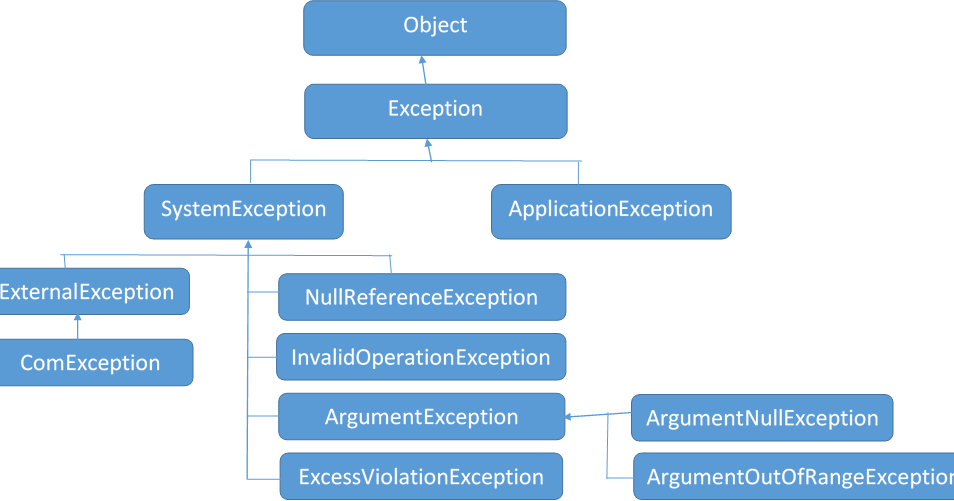
Exception cung cấp một cách để truyền điều khiển từ một phần của một chương trình tới phần khác. Xử lý ngoại lệ trong C# được xây dựng dựa trên 4 từ khóa là: try, catch, finally, và throw.

* **try**: Một khối try nhận diện một khối code mà ở đó các exception cụ thể được kích hoạt. Nó được theo sau bởi một hoặc nhiều khối catch.
* **catch**: Một chương trình bắt một Exception với một Exception Handler tại vị trí trong một chương trình nơi bạn muốn xử lý vấn đề đó. Từ khóa catch trong C# chỉ dẫn việc bắt một **exception**.
* **finally**: Một khối finally được sử dụng để thực thi một tập hợp lệnh đã cho, khối lệnh finally luôn luôn được thực thi dù có hay không một exception đươc ném hoặc không được ném. Ví dụ, nếu bạn mở một file, nó phải được đóng, nếu không sẽ có một exception được tạo ra.
* **throw**: Một chương trình ném một exception khi có một vấn đề xuất hiện. Điều này được thực hiện bởi sử dụng từ khóa throw trong C#.

## Cú pháp xử lý



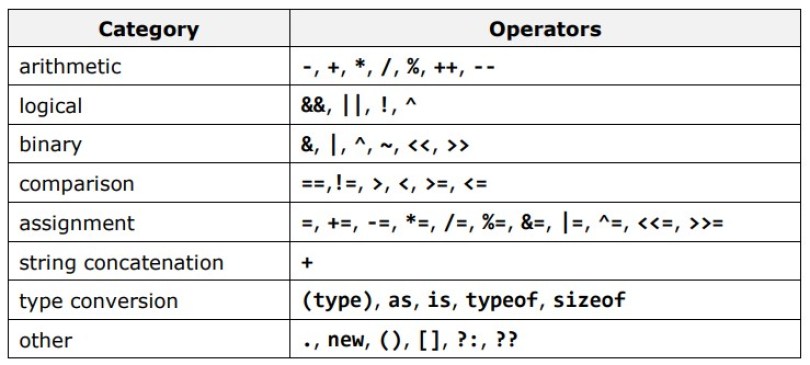
## Các lớp Exception của hệ thống



## Ví dụ

# Bài 16: Toán tử (operators)

* Toán tử được định nghĩa như sau:
  + Là một công cụ để thao tác với dữ liệu.
  + Một toán tử là một ký hiệu dùng để đại diện cho một thao tác cụ thể được thực hiện trên dữ liệu.
* Có 7 loại toán tử cơ bản:
  + Toán tử số học.
  + Toán tử logic.
  + Toán tử nhị phân.
  + Toán tử so sánh.
  + Toán tử gán.
  + Toán tử nối.
  + Toán tử chuyển đổi
  + Toán tử khác



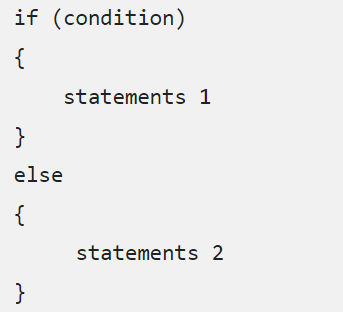
Tham khảo: <https://docs.microsoft.com/vi-vn/dotnet/csharp/language-reference/operators/>

# Bài 17: Cấu trúc điều khiển if else

Cấu trúc điều kiện là cấu trúc rẽ nhánh cho phép phân tách việc thực thi code thành nhiều hướng khác nhau tuỳ thuộc vào một điều kiện nào đó. Điều kiện này thông thường được xác định theo giá trị của biến hoặc biểu thức.

C# sử dụng 2 cấu trúc điều kiện là if-else và cấu trúc switch-case.

* Condition: Là một biểu thức hoặc 1 giá trị trả về dạng true/false.
* Statement 1: Là các lệnh sẽ được thực thi nếu condition có giá trị là true
* Statement 2: Là các lệnh sẽ được thực thi nếu condition có giá trị là false.



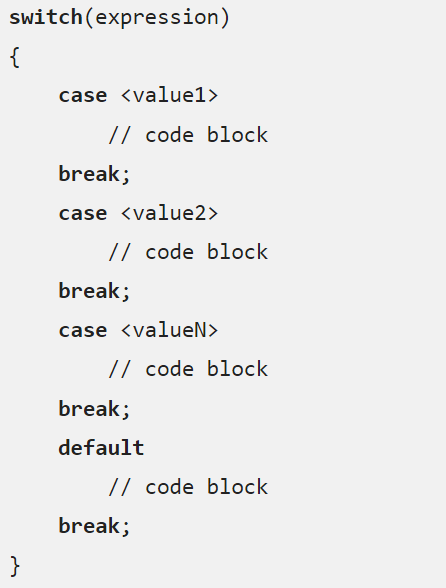
Một số lưu ý khi sử dụng if-else:

* Nếu statement 1 hoặc statement 2 chỉ có 1 lệnh duy nhất thì có thể không cần cặp ngoặc {}
* Nhánh else {} không bắt buộc, if thì bắt buộc phải có
* Bình thường bạn chỉ có thể tạo ra 2 nhánh rẽ: 1 nhánh if và 1 nhánh else.
* Để taoh thêm nhiều nhánh rẽ nữa, bạn có thể kết hợp thêm các nhánh else if vào cấu trúc trên. Số lượng else if không giới hạn.
* Bạn có thể lồng nhiều if-else với nhau.

# Bài 18: Cấu trúc điều khiển Swicth case

Ở bài trước chúng ta đã tìm hiểu cấu trúc rẽ nhánh if-else. Cấu trúc này chỉ cho phép rẽ tới 2 nhánh. Nếu muốn rẽ nhiều nhánh bạn phải lồng ghép các nhánh else-if khiến code trở nên khó đọc.

C# cung cấp một cấu trúc khác để thực hiện rẽ nhiều nhánh thay cho việc lồng ghép nhiều if-else là cấu trúc switch-case.



# Bài 19: Vòng lặp (loop)

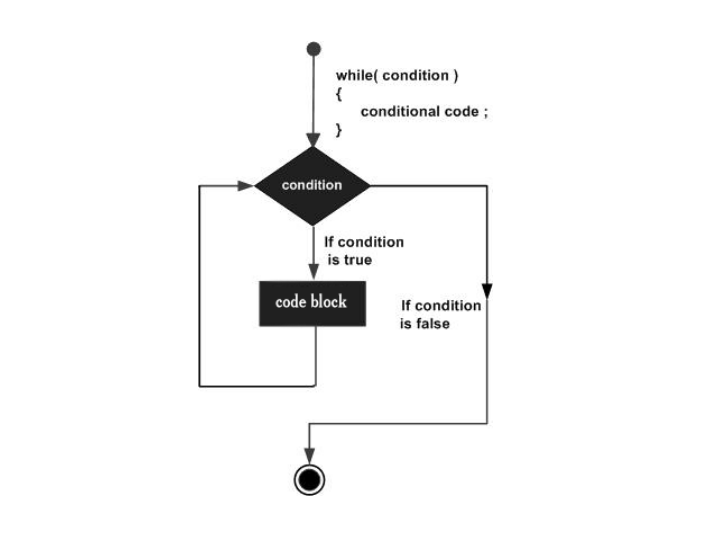
Trong thực tế khi bạn cần thực thi một khối lệnh nhiều lần. Vòng lặp cho phép chúng ta thực thi một câu lệnh hoặc một khối lệnh nhiều lần.

## Các kiểu vòng lặp

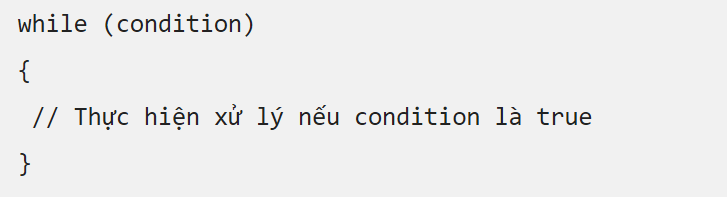
|  |  |
| --- | --- |
| **Kiểu vòng lặp** | **Mô tả** |
| While | Lặp lại một hoặc một nhóm các lệnh trong khi điều kiện đã cho là đúng. Nó kiểm tra điều kiện trước khi thực thi thân vòng lặp |
| For | Thực thi một dãy các lệnh nhiều lần và tóm tắt đoạn code mà quản lý biến vòng lặp |
| Do-while | Giống lệnh while ngoại trừ điểm là nó kiểm tra điều kiện ở cuối thân vòng lặp và luôn thực hiện vòng lặp đầu tiên dù điều kiện đúng hay không. |
| Foreach | Được sử dụng để duyệt lần lượt từng phần tử trong một tập hợp, mảng có sẵn |
| Vòng lặp lồng nhau | Bạn có thể sử dụng một hoặc nhiều vòng lặp trong các vòng lặp while, for hoặc do..while khác. |

## Vòng lặp while

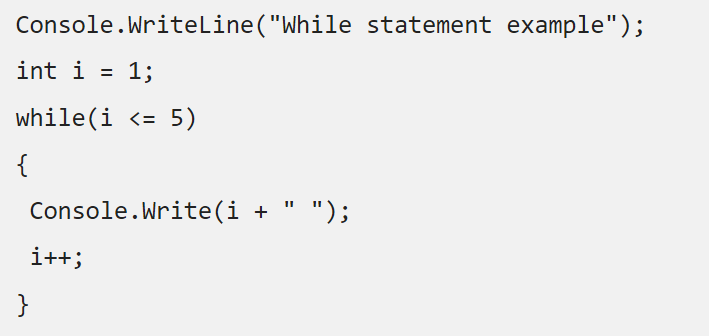
Nếu biểu thức điều kiện theo sau while là đúng (true) thì khối lệnh bên trong vòng lặp được thực hiện và sau mỗi lần lặp biểu thức điều kiện được kiểm tra lại và nếu biểu thức điều kiện là sai (false) vòng lặp sẽ kết thúc.



Cú pháp:

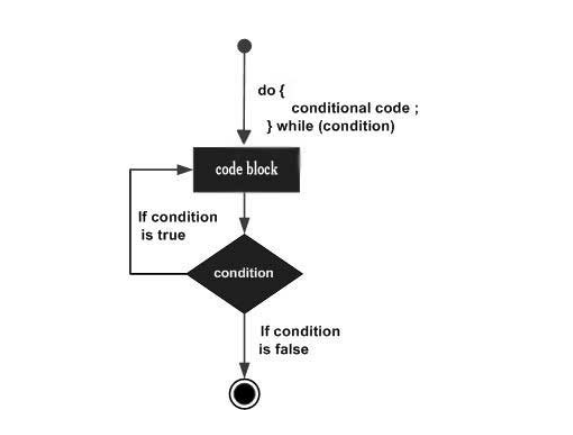


Ví dụ:

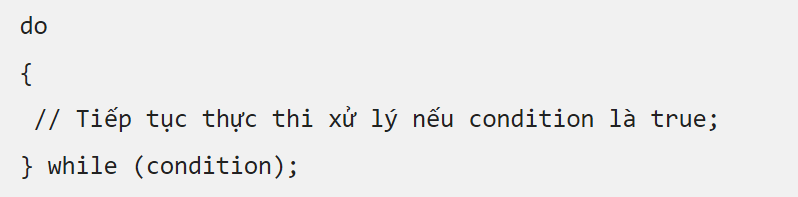


## Vòng lặp do…while

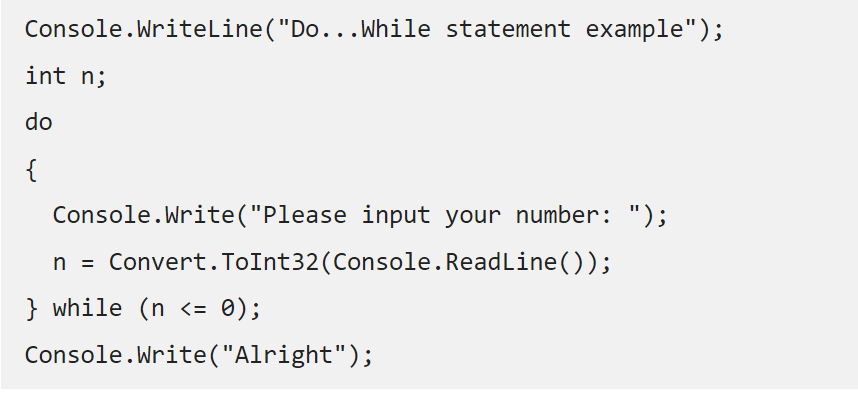
Vòng lặp do…while cũng tương tự như vòng lặp while tuy nhiên nó luôn luôn thực thi khối lệnh bên trong ít nhất một lần vì vòng lặp do…while kiểm tra điều kiện lặp ở cuối. Đó chính là điểm khác biệt duy nhất giữa vòng lặp while và vòng lặp do…while



Cú pháp:

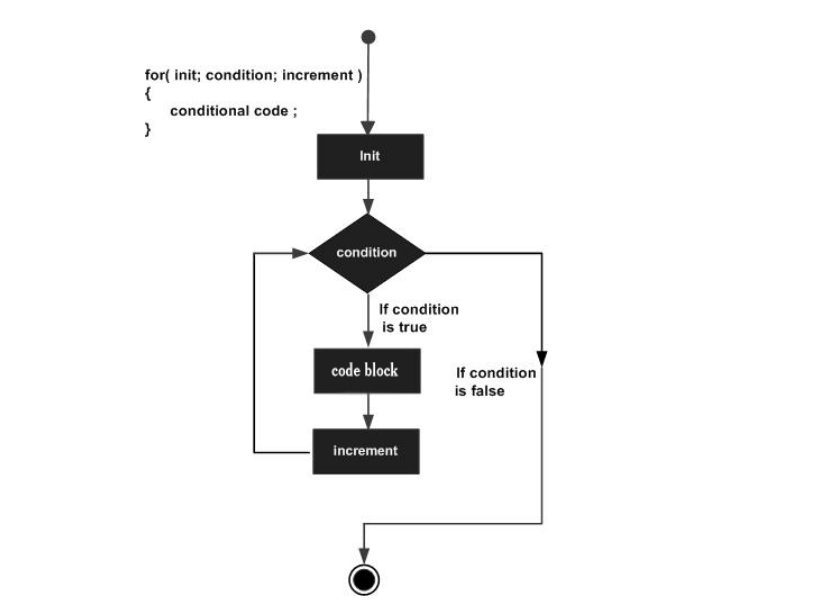


Ví dụ:

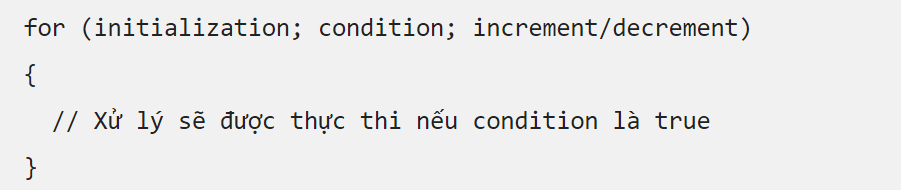


## Vòng lặp for

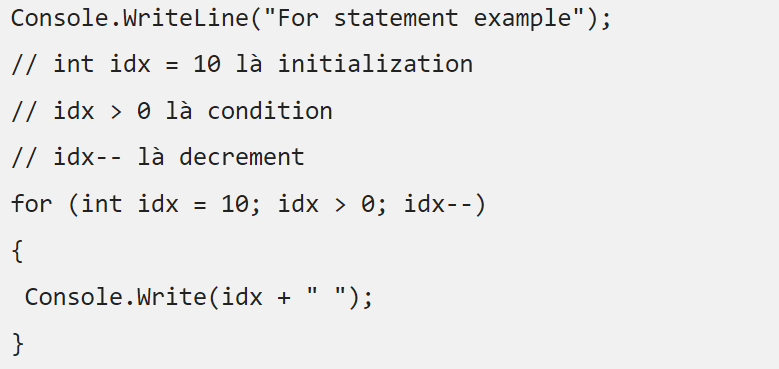
Tương tự như vòng lặp while, những câu lệnh bên trong vòng lặp for sẽ được thực thi nếu biểu thức điều kiện là đúng (true).



Cú pháp:

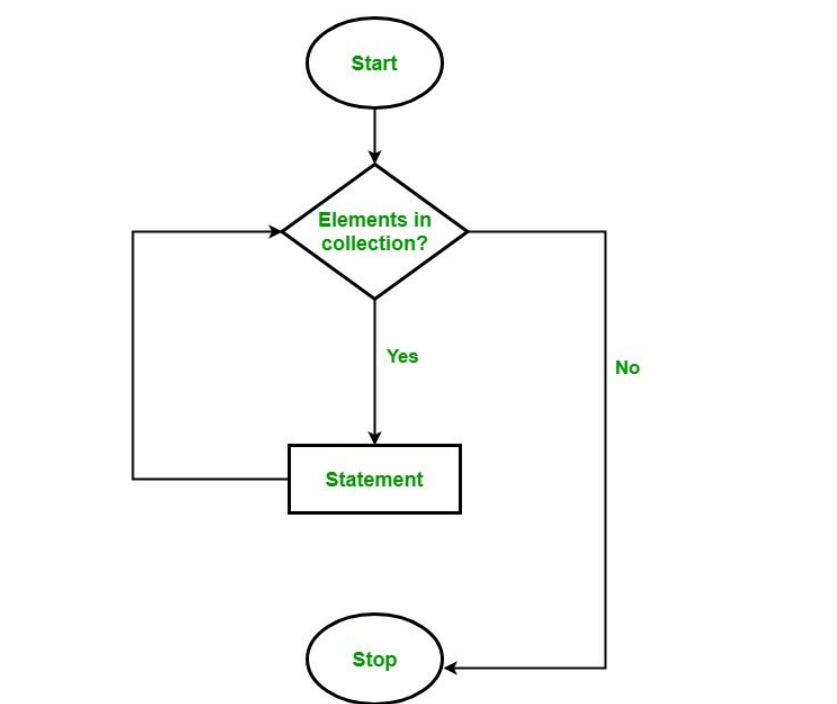


Ví dụ:

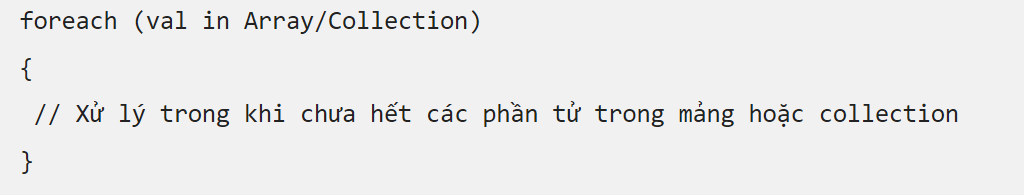


## Vòng lặp foreach

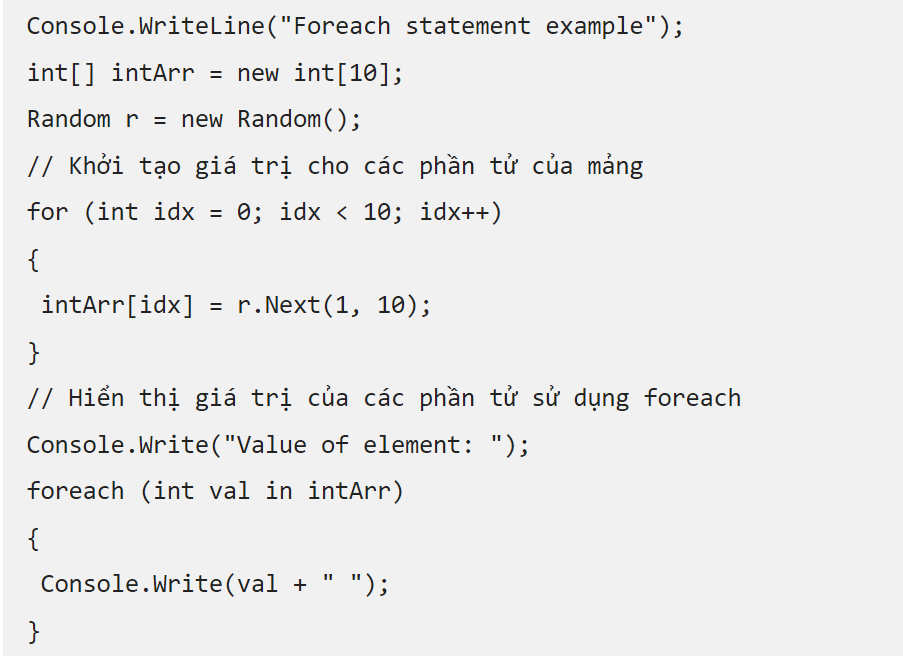
Vòng lặp foreach thường được sử dụng để xử lý trên mảng hoặc trên collection để truy cập giá trị của các phần tử trong mảng hoặc collection.



Cú pháp:

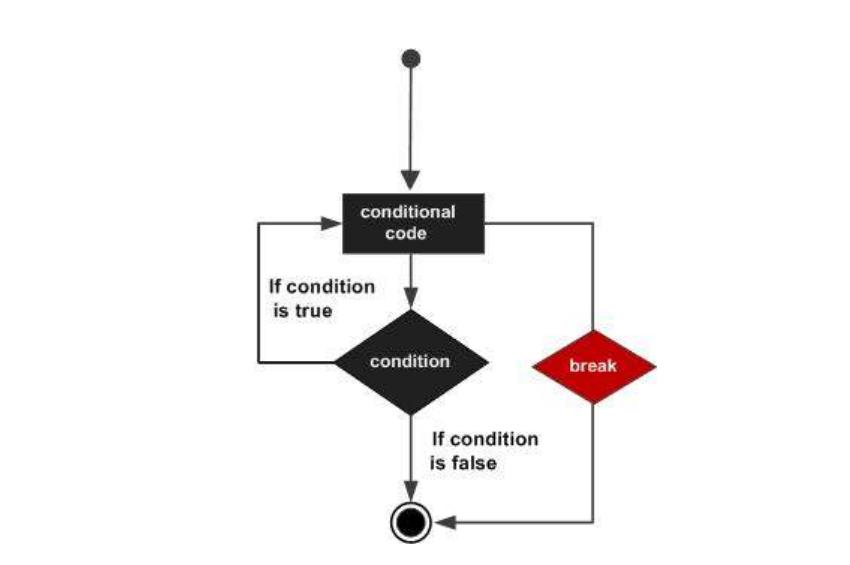


Ví dụ:

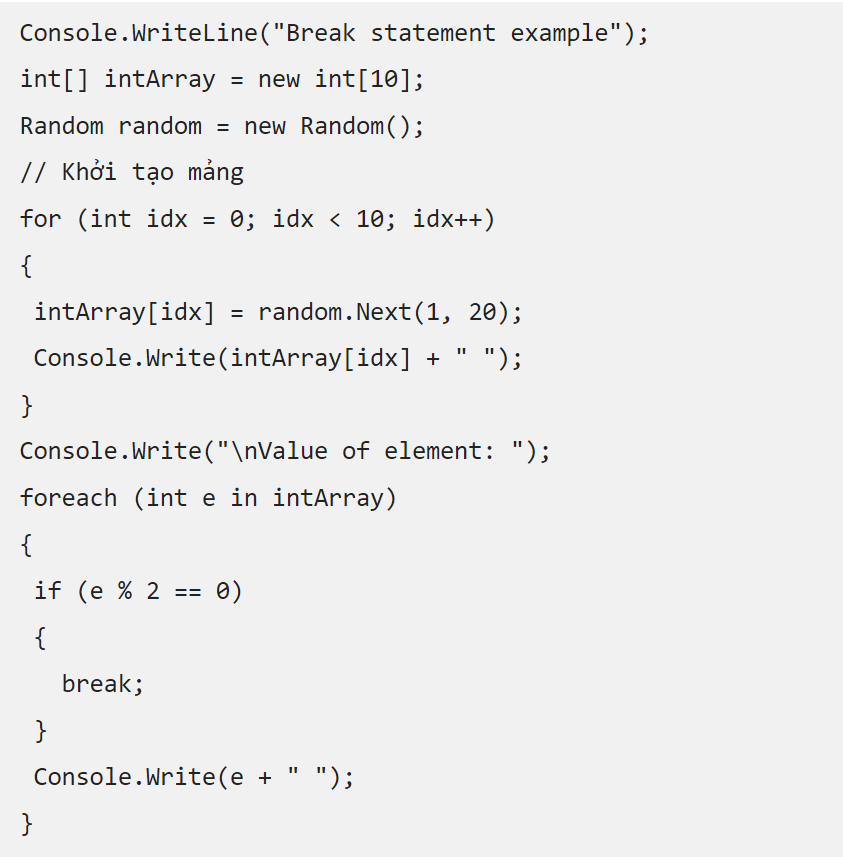


## Câu lệnh break

Như cấc bạn đã biết, câu lệnh break được sử dụng trong switch…case để kết thúc switch…case. Trong vòng lặp câu lệnh break được sử dụng để kết thúc vòng lặp.

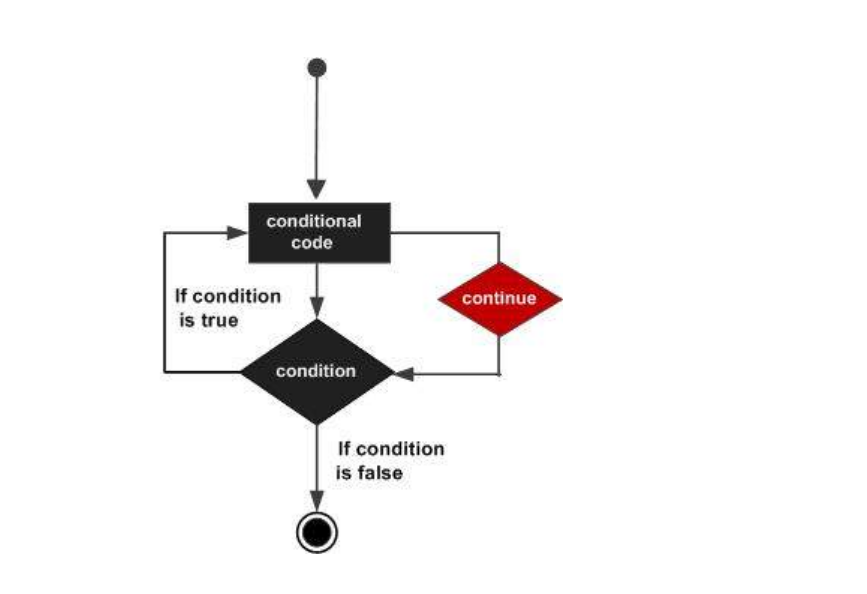


Ví dụ sử dụng câu lệnh break để kết thúc vòng lặp nếu có một phần tử nào đó chia hết cho 2:

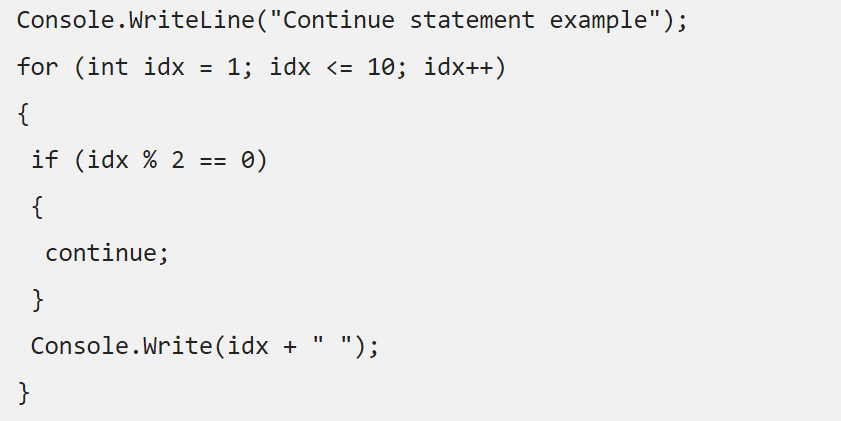


## Câu lệnh continue

Nếu như câu lệnh break sẽ kết thúc vòng lặp thì câu lệnh continue sẽ bỏ qua những xử lý ở sau câu lệnh continue.



Ví dụ sử dụng câu lệnh continue để hiển thị các số lẻ từ 1 đến 10:



# Bài 20: Bài tập kết thúc chương 2

Bài tập 1: Viết chương trình giải phương trình bậc nhất (ax+b=0) với a,b nhập vào từ bàn phím.

Bài tập 2: Viết phương trình giải phương trình bậc 2 (ax2 + bx +c = 0) với a,b,c nhập vào từ bàn phím.

Bài tập 3: Tính giai thừa của một số tự nhiên bất kỳ nhập vào từ bàn phím.

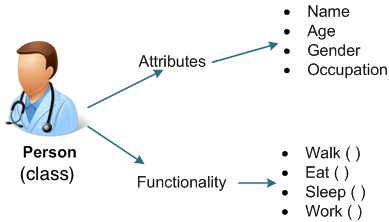
Bài tập 4: Tìm số lượng số nguyên tố từ 1 đến n. Với n nhập từ bàn phím

Bài tập 5: Tìm dãy số fibonancy thứ n, với n nhập từ bàn phím.

# Bài 21: Lập trình hướng đối tượng

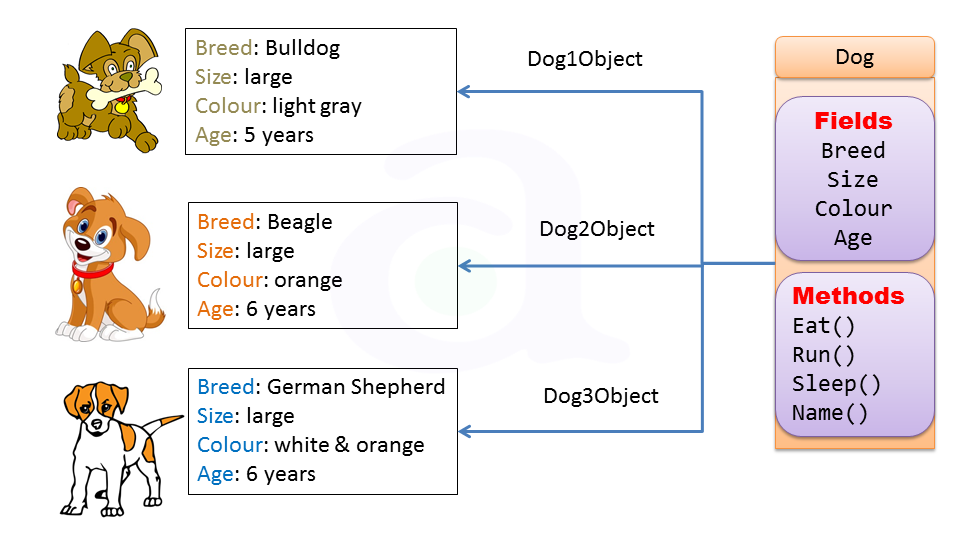
Lập trình hướng đối tượng (Object Oriented Programing) hay còn gọi là OOP. Là một kỹ thuật lập trình cho phép các lập trình viên có thể ánh xạ các thực thể bên ngoài đời thực và trừu tượng hoá thành các class và object trong mã nguồn.

Trong đó: Mỗi thực thể được ánh xạ thành các class có chứa các thông tin mô tả của thực thể đó (gọi là thuộc tính) và các hành động của thực thể (gọi là các phương thức).

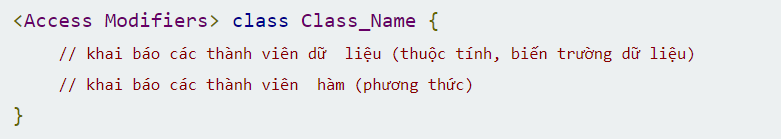


## Giới thiệu về Class và Object

* Một Class là một Blueprint (kế hoạch) hay Prototype (nguyên mẫu) xác định biến và các phương thức (hay function) chung với tất cả các đối tượng cùng loại.
* Một Object (đối tượng) là một cụ thể, thể hiện của một Class.

Các đối tượng thường được dùng để mô tả đối tượng trong thế giới thực mà bạn thấy hàng ngày.

Cú pháp khai báo lớp:



## Sử dụng Constructor (1 constructor và nhiều constructor)

Phương thức khởi tạo (Constructor) là những phương thức đặc biệt được gọi đến ngay khi khởi tạo 1 đối tượng nào đó.

Đặc điểm

* Có tên trùng với tên lớp.
* Không có kiểu trả về.
* Được tự động gọi khi 1 đối tượng thuộc lớp được khởi tạo.
* Nếu như bạn không khai báo bất kỳ phương thức khởi tạo nào thì hệ thống sẽ tự tạo ra phương thức khởi tạo mặc định không đối số và không có nội dung gì.
* Có thể có nhiều constructor bên trong 1 lớp.

## Phạm vi truy cập (Access Modifiers)

Phạm vi truy cập là cách mà người lập trình quy định về quyền được truy xuất đến các thành phần của lớp.

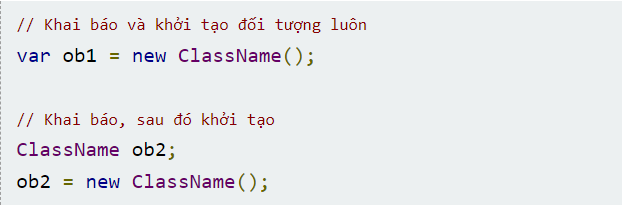
Trong C# có 5 phạm vi truy cập:

1. public: không giới hạn phạm vi truy cập
2. protected: chỉ truy cập trong nội bộ lớp hay các lớp kế thừa
3. private: chỉ truy cập được từ các thành viên của lớp chứa nó
4. internal: chỉ truy cập được trong cùng assembly (dll, exe)
5. protected internal: truy cập được khi cùng assembly hoặc lớp kế thừa

Lưu ý:

* Nếu khai báo lớp mà không chỉ ra phạm vi truy cập thì mặc định là **internal**
* Nếu khai báo thành phần bên trong lớp mà không chỉ ra phạm vi cụ thể thì mặc định là **private**

## Tạo và sử dụng đối tượng



Sau khi đối tượng lớp (object) được tạo, bạn có thể truy cập đến các thuộc tính, trường dữ liệu và phương thức của đối tượng đó bằng ký hiệu . theo quy tắc **object.tên\_thuộc\_tính** hay **object.tên\_phương\_thức**

## Từ khoá this

Từ khóa **this** dùng trong các phương thức của lớp, nó tham chiếu đến đối tượng hiện tại sinh ra từ lớp. Sử dụng **this** để tường minh, tránh sự không rõ ràng khi truy cập thuộc tính, phương thức hoặc để lấy đối tượng lớp làm tham số cho các thành phần khác ...

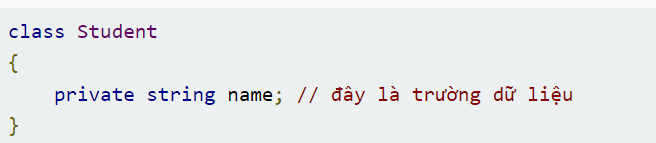
## Thuộc tính (Properties)

### Trường dữ liệu của lớp

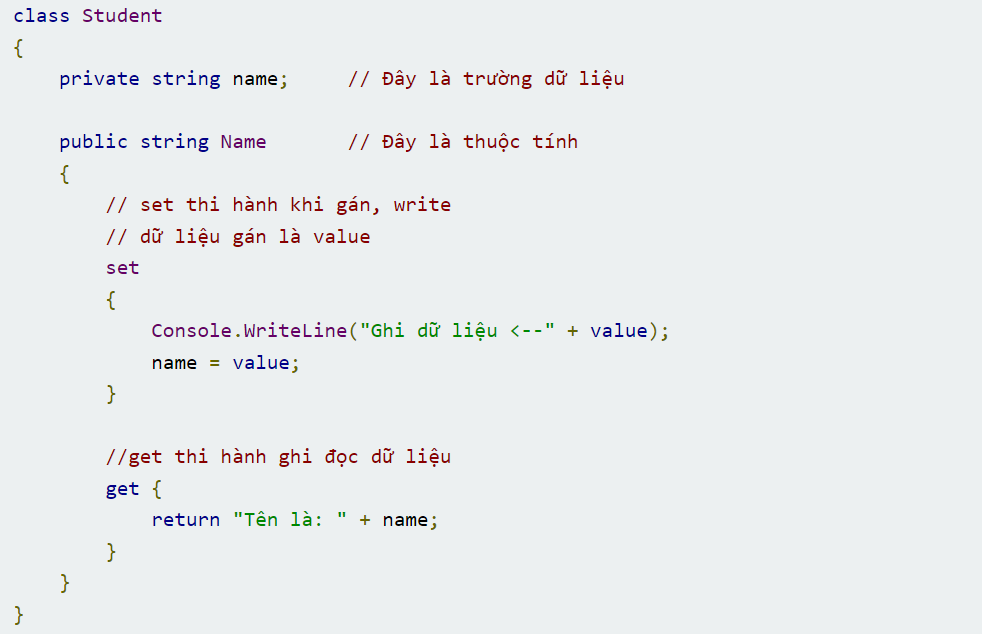
Trường dữ liệu - khai báo như biến trong lớp, nó là thành viên của lớp, nó là biến. Trường dữ liệu có thể sử dụng bởi các phương thức trong lớp, hoặc nếu là public nó có thể truy cập từ bên ngoài, nhưng cách hay hơn để đảm bảo tính đóng gói khi cần truy cập thuộc tính hãy sử dụng phương thức, còn bản thân thuộc tính là private. Chúng ta đã sử dụng các trường dữ liệu ở những ví dụ trên.

### Thuộc tính, bộ truy cập accessor setter/getter

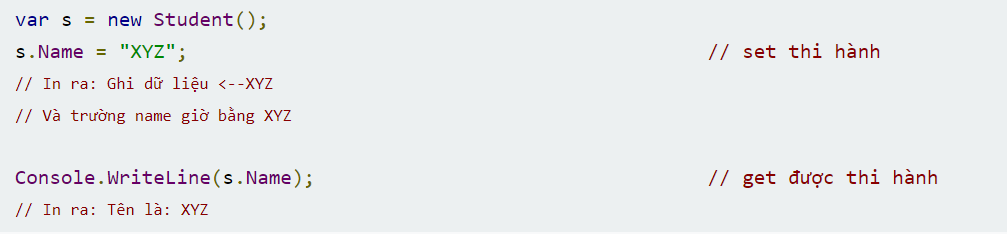
Ngoài cách sử dụng trường dữ liệu, khai báo như biến ở phần trước, khai báo THUỘC TÍNH tương tự nhưng nó có cơ chế accessor (bộ truy cập), một cơ chế hết sức linh hoạt khi bạn đọc / ghi dữ liệu vào thuộc tính. Hãy tìm hiểu qua một ví dụ sau:



Lớp này có một trường dữ liệu private là **name**. Giờ ta sẽ khai báo một thuộc tính có tên **Name** với modify là **public**, thuộc tính này khi đọc sẽ thi hành một đoạn code gọi là **get**, khi ghi (gán) dữ liệu nó thi hành đoạn code gọi là **set**, thuộc tính **Name** sẽ phối hợp cùng trường dữ liệu **name**



Khi thực hiện:



Trong C#, phương thức truy xuất và phương thức cập nhật đã được nâng cấp lên thành 1 cấu trúc mới ngắn gọn hơn và tiện dụng hơn đó là **property**.

Sử dụng property giúp ta có thể thao tác dữ liệu tự nhiên hơn nhưng vẫn đảm bảo tính đóng gói của lập trình hướng đối tượng.

Lưu ý:

* Người ta dùng Property thay cho phương thức truy vấn, phương thức cập nhật vì thế tên property thường phải làm gợi nhớ đến tên thuộc tính private bên trong lớp.
* Tuỳ theo nhu cầu và tính bảo mật mà người lập trình có thể ngăn không cho gán giá trị hoặc ngăn không cho lấy dữ liệu bằng cách bỏ đi từ khoá tương ứng.
* Thuộc tính accessor có thể khai báo thiếu **set** hoặc **get**, nếu thiếu **set** nó trở thành loại chỉ đọc (readonly). Sử dụng **set** rất tiện lợi cho thao tác kiểm tra tính hợp lệ của dữ liệu khi gán, hoặc tự động thực hiện một số tác vụ mỗi khi dữ liệu được gán.

## Tìm hiểu tính đóng gói lập trình hướng đối tượng

Tính đóng gói mục đích hạn chế tối đa việc can thiệp trực tiếp vào dữ liệu, hoặc thi hành các tác vụ nội bổ của đối tượng. Nói cách khác, một đối tượng là hộp đen đối với các thành phần bên ngoài, nó chỉ cho phép bên ngoài tương tác với nó ở một số phương thức, thuộc tính, trường dữ liệu nhất định - hạn chế.

C# triển khai tính đóng gói này chính là sử dụng các Access Modifiers: public private protected internal khi khai báo lớp, phương thức, thuộc tính, trường dữ liệu (biến).

* **public** thành viên có thể truy cập được bởi code bât kỳ đâu, ngoài đối tượng, không có hạn chế truy cập nào.
* **private** phương thức, thuộc tính, trường khai báo với private chỉ có thể truy cập, gọi bởi các dòng code cùng lớp.
* **protected** phương thức, thuộc tính, trường chỉ có thể truy cập, gọi bởi các dòng code cùng lớp hoặc các lớp kế thừa nó.
* **internal** truy cập được bởi code ở cùng assembly (file).
* **protected internal** truy cập được từ code assembly, hoặc lớp kế thừa nó ở assembly khác.

# Bài 22: Sử dụng mảng (Arrays)

## Giới thiệu về mảng

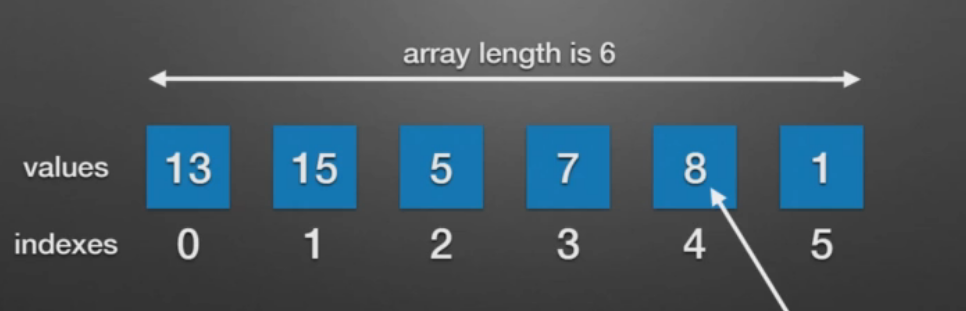
Mảng trong C# là một cấu trúc dữ liệu lưu trữ một dãy các phần tử có bộ nhớ nằm liên tiếp nhau và có kích thước cố đinh.

Mỗi phần tử trong mảng được truy cập thông qua chỉ số. Phần tử đầu tiên có chỉ số là 0 và phần tử cuối cùng có chỉ số n – 1 (trong đó n là số lượng phần tử có trong mảng).

Dựa vào cách mảng lưu trữ các phần tử, mảng có thể được phân thành 2 loại là mảng một chiều (Single-dimensional Arrays) và mảng đa chiều (Multi-dimensional Arrays).

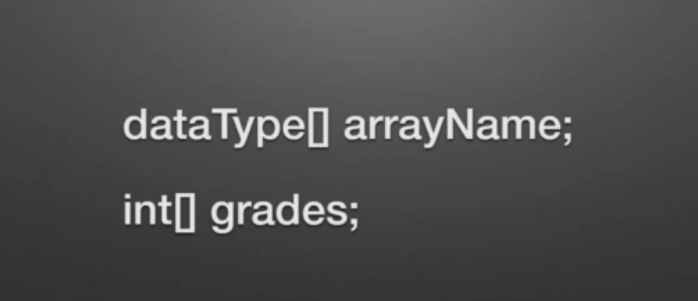
Tính chất:

* Độ dài cố định
* Chỉ lưu các phần tử cùng kiểu dữ liệu
* Lưu trữ và cấp phát các ô nhớ liên tiếp nhau

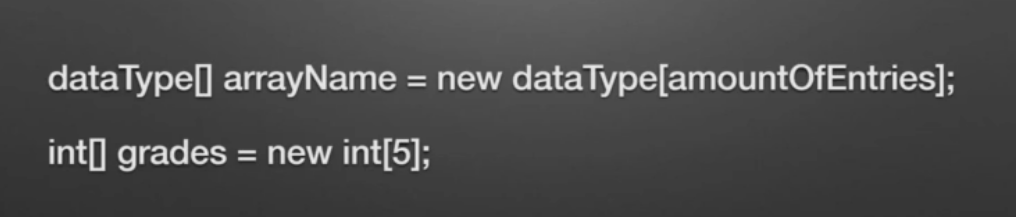


## Khai báo và khởi tạo mảng

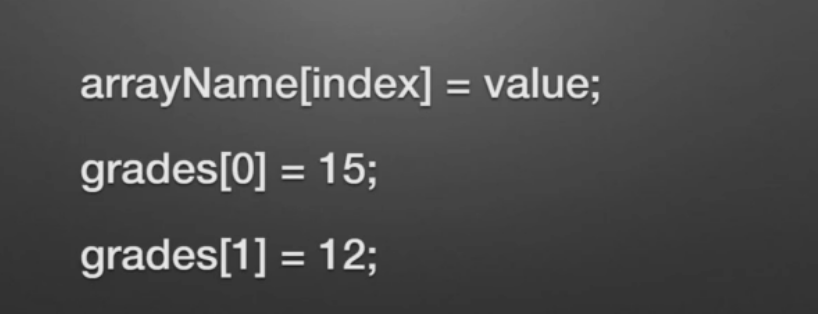
### Khai báo



### Khai báo và khởi tạo



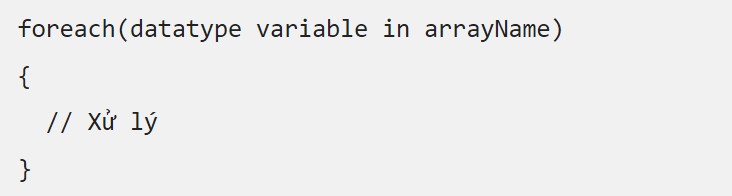
### Gán giá trị cho một mảng



### Thực hành tạo mảng và truy xuất giá trị

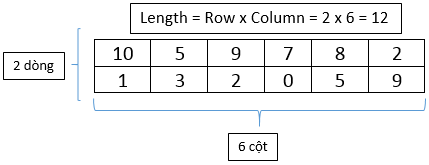
## Lặp qua mảng

* Sử dụng vòng lặp for hoặc foreach



## Mảng 2 chiều

Không giống như mảng một chiều, mảng đa chiều cho phép chúng ta lưu trữ dữ liệu trên nhiều dòng. Kích thước của mảng được xác định dựa vào số dòng và số cột tương tự như một sheet trong Microsoft Excel và được phân làm 2 loại như sau

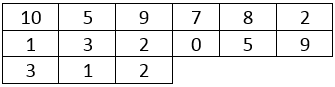


Khai báo mảng 2 chiều:



### Jagged Array

Jagged Array tương tự Rectangular Array ngoại trừ số cột trên mỗi dòng có thể khác nhau. Hình bên dưới là một ví dụ

[](https://giasutinhoc.vn/wp-content/uploads/2016/06/su-dung-mang-trong-c-sharp-3.png)

# Bài 23: Tổng quan về Generic và Non-Generic Collection

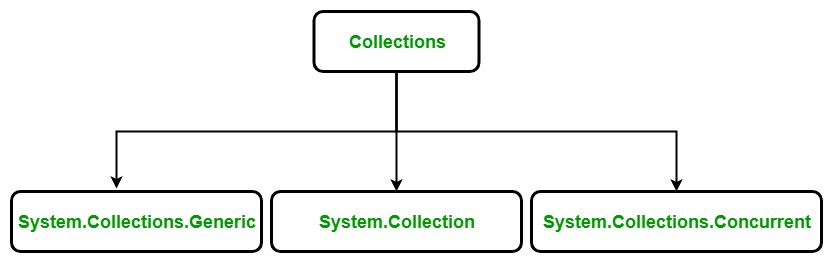
* Collection là các class mà chúng ta có thể dùng để lưu trữ tập hợp các object.
* Không giới hạn một kiểu của object
* Không giới hạn kích thước, nó có thể tăng lên khi chúng ta thêm phần tử.

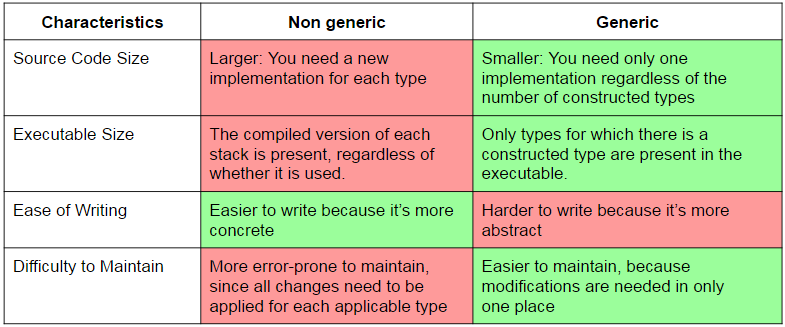
Tại sao chúng ta cần đến Collections:

* Sử dụng chúng để lưu trữ, quản lý và thao tác một nhóm các đối tượng hiệu quả hơn.

Có 2 loại Collections:

|  |  |
| --- | --- |
| **Non-Generic** | **Generic** |
| Có thể lưu trữ bất cứ kiểu nào của đối tượng | Giới hạn một kiểu của đối tượng |
| Nằm trong namespace System.Collections | Nằm trong System.Collections.Generic |





# Bài 24: Các loại Collection

## Lists

## Dictionaries

## Queue & Stack

Tham khảo: <https://tedu.com.vn/lap-trinh-c/cnet-can-ban-hieu-biet-ve-cac-collection-trong-net-framework-90.html>

# Bài 25: Cách debug ứng dụng C#

# Bài 26: Lập trình hướng đối tượng (OOP)

# Bài 27: Tính chất kế thừa (Inheritance)

## Giới thiệu về tính chất kế thừa

## Ví dụ demo

## Từ khoá virtual và override

# Bài 28: Tìm hiểu về Interface

# Bài 29: Giới thiệu về IEnumerator và IEnumberable

# Bài 30: Tính đa hình (Polymorphism)

## Giới thiệu tính đa hình

## Từ khoá sealed

## Abstract class

## Interface và abstract class

## Ví dụ về đọc ghi 1 file

# Bài 31: Tính trừu tượng (Abstraction)

# Bài 32: Sử dụng kiểu tập hợp (Enum)

# Bài 33: Sử dụng Math class

# Bài 34: Sử dụng DateTime

# Bài 35: Kiểu Nullable

# Bài 36: Hiểu và sử dụng Lamda Expression

# Bài 37: Sử dụng Multithreads

# Bài 38: Tổng kết khoá học

## Tổng kết những vấn đề đã học

## Các vấn đề cần tìm hiểu thêm (chưa nằm trong khoá học)

1. Regex
2. Struct
3. Delegate & Event
4. Anonymous Methods