Contents

[Kiểu dữ liệu 2](#_Toc477955370)

[1. Kiểu cơ bản (primitive type) 2](#_Toc477955371)

[2. Kiểu var 3](#_Toc477955372)

[3. Kiểu dynamic 3](#_Toc477955373)

[Tầm vực (Access modifiers) 3](#_Toc477955374)

[1. public 3](#_Toc477955375)

[2. private 3](#_Toc477955376)

[3. protected 3](#_Toc477955377)

[4. internal 3](#_Toc477955378)

[5. protected internal 3](#_Toc477955379)

[Các hàm trong C# 4](#_Toc477955380)

[OOP trong C# 6](#_Toc477955381)

[Sealed class là gì? 6](#_Toc477955382)

[Ví dụ về sealed class 6](#_Toc477955383)

# Kiểu dữ liệu

## Kiểu cơ bản (primitive type)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Kiểu trong C# | Số byte | Giá trị mặc định | Giá trị nhỏ nhất | Giá trị lớn nhất | Kiểu .NET tương ứng | Chú thích |
| Kiểu Integer (Integer type) | | | | | | |
| byte | 1 | 0 | 0 | 255 | Byte |  |
| sbyte | 1 | 0 | -27 | 27 - 1 | SByte |  |
| short | 2 | 0 | -215 | 215 - 1 | Int16 |  |
| ushort | 2 | 0 | 0 | 216 | UInt16 |  |
| int | 4 | 0 | -231 | 231 – 1 | Int32 |  |
| uint | 4 | 0u | 0 | 232 | UInt32 |  |
| long | 8 | 0L | -263 | 263 - 1 | Long |  |
| ulong | 8 | 0u | 0 | 264 | ULong |  |
| Kiểu dấu chấm động (floating-point) | | | | | | |
| float | 4 | 0.0f | ±1.5×10-45 | ±3.4×1038 | Float | Độ chính xác 7 chữ số |
| double | 8 | 0.0d | ±5.0×10-324 | ±1.7×10308 | Double | Độ chính xác 15, 16 chữ số |
| decimal | 8 | 0.0m | -7.9×1028 | +7.9×1028 | Decimal | Hoàn toàn chính xác |
| Các kiểu khác | | | | | | |
| bool | 1 | false | 2 giá trị: true, false | | Boolean |  |
| char | 2 | ‘\u0000’ | ‘\u0000’ |  | ‘\uffff’ |  |
| string | null |  |  |  | String |  |
| object | null |  |  |  | Object |  |

## Kiểu var

* Compiler sẽ tự động xác định kiểu của biến (implicitly-typed), khác với việc chúng ta xác định một cách tường minh (explicitly-typed)
* Phải được khởi tạo ngay lúc gán
* Có thể dùng để tạo một kiểu vô danh (anonymous type)

## Kiểu dynamic

* Một object được khai báo kiểu dynamic sẽ được bỏ qua việc kiểm tra của compiler vào compile-time và xác định kiểu vào thời điểm run-time.

# Mảng

Mảng một chiều

//Lúc này a vẫn null

int[] a;

//Khởi tạo mảng b trong bộ nhớ heap với độ dài 6 phần tử

int[] b = new int[6];

//Khởi tạo mảng c với 4 phần tử cho trước

int[] c = new int[] { 1, 2, 3, 4 };

//Khởi tạo mảng d với 5 phần tử cho trước

int[] d = { 1, 2, 3, 4, 5 };

//Khởi tạo mảng e với 1000 phần tử có giá trị = 1  
int[] e = Enumerable.Repeat<int>(1, 1000).ToArray();

Mảng hai chiều

|  |
| --- |
| //Khai báo mảng hai chiều với 2 hàng và 3 cột  int[,] a = new int[2, 3];  private void createArrayA()  {  for (int i = 0; i < 2; i++)  {  for (int j = 0; j < 3; j++)  {  a[i, j] = i \* 3 + j + 1;  }  }  /\*Kết quả: a = [  1, 2, 3  4, 5, 6  ];  \*/  }  //Tạo mảng với dữ liệu cho trước  int[,] b = { { 1, 2 }, { 1, 3 } };  //Lỗi  int[,] c = { new int[] { 1, 2 }, new int[] { 2, 3 } }; |

Mảng của mảng (jagged array)

|  |
| --- |
| //Khai báo  //Lưu ý: chúng ta chỉ quy định được chiều dài của mảng chính chứ //không quy định được chiều dài các mảng con do chưa biết chiều dài chính //xác của các mảng con  int[][] a = new int[3][];  //Tạo giá trị cho mảng a  private void createArrayA()  {  for (int i = 0; i < 3; i++)  {  a[i] = new int[] { i \* 3 + 1, i \* 3 + 2, i \* 3 + 3 };  }  /\*Kết quả: a = [  \* [1, 2, 3],  \* [4, 5, 6],  \* [7, 8, 9],  ];  \*/  }  //Khai báo với các giá trị cho trước kiểu bá đạo  //Yêu cầu: do khai báo mảng b là int[3][], chiều dài mảng trước  // đã được quy định là 3 nên trong mảng chính phải tạo  // đúng 3 mảng con  int[][] b = new int[3][] {  new int[] { 1, 2, 3 },  new int[] { 1, 2},  new int[] { 1, 2, 3, 4}  };  //Khai báo mảng mà không cần biết chiều dài của mảng chính  int[][] c = {  new int[] { 1, 2, 3},  new int[] { 1, 2},  }; |

# 

# Tầm vực (Access modifiers)

## public

Có thể được truy cập trong cùng assembly hoặc những assembly khác có reference tới.

## private

Chỉ có thể được truy cập từ code trong cùng một class hoặc struct

## protected

Có thể được truy cập từ code trong cùng class, struct hoặc lớp con.

## internal

Có thể được truy cập trong cùng một assembly nhưng không thể truy cập từ assembly khó dù có reference

## protected internal

Có thể được truy cập trong cùng một assembly hoặc từ class con của assembly khác

Khi không có access modifier, access modifier mặc định sẽ được gọi

|  |  |  |
| --- | --- | --- |
| Loại | Mặc định | Access modifiers cho phép khai báo |
| enum | public | không có |
| class | private | 5 loại trên |
| interface | public | không có |
| struct | private | public, internal, private |

# Các hàm trong C#

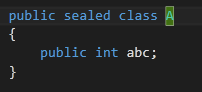
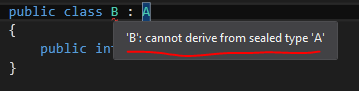
# if…else

# OOP trong C#

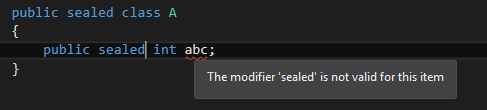
## Sealed class là gì?

Sealed class được dùng để chặn việc được thừa kế của class. Từ khóa “sealed” có thể được dùng với cả method để bảo đảm rằng method sẽ không được thừa kế ở những class khác. Xem [ví dụ về sealed class](#_Ví_dụ_về).

## Ví dụ về sealed class

Tuy nhiên, class A vẫn có thể thừa kế class B



Từ khóa “sealed” không được dùng cho biến

Index

P

Primitive

Nguồn tham khảo

Kenhsinhvien.net - Giáo trình C#

Fundametal of computer programming with C#

http://stackoverflow.com/documentation/c%23/topics