# CODING STYLE: C# (Google style)

* Đây là hướng dẫn cho lập trình C# trong nội bộ Google, và là quy tắc cơ bản cho lập trình C# ở Google.

## Quy tắc đặt tên

|  |  |  |
| --- | --- | --- |
| Tên quy tắc | Mô tả | Ví dụ |
| Tên lớp, phương thức, kiểu liệt kê, các trường/thuộc tính public, namespaces | - Viết hoa chữ cái đầu. | **public** **enum** MyEnum {  Yes,  No,  } |
| Quy tắc đặt tên biến toàn cục | - Chữ cái đầu tiên trong từ định danh là chữ thường và chữ cái đầu tiên của mối từ nối theo sau phải được viết hoa | private int camelCase |
| Quy tắc đặt tên trường, thuộc tính | - Có gạch chân ở đầu | private int \_camelCase |
| Quy ước đặt tên không bị ảnh hưởng bởi “Quyền truy cập” như Const, static, readonly,etc.. | - Có gạch chân giữa các từ, uppercase | public static readonly string TEN\_HANG\_SO |
| Đối với quy ước viết hoa/thường (casing), một "từ" được hiểu là bất kỳ chuỗi ký tự nào không chứa khoảng trống bên trong, bao gồm cả từ viết tắt |  | MyRpc: true  MyRPC: false |
| Tên Interface | Bắt đầu bằng I | Interface IClass |
| Tên file và thư mục | - Viết hoa chữ cái đầu - Chỉ có 1 lớp chính trong một file (có tên cùng với tên file) | MyFile.cs  Class MyFile |

## Quy tắc tổ chức

|  |  |  |
| --- | --- | --- |
| Tên quy tắc | Mô tả | Ví dụ |
| Quyền truy cập | Theo trình tự: public protected internal private new abstract virtual override sealed static readonly extern unsafe volatile async | public static override void Ham() |
| Từ khóa “Using” | Định nghĩa từ khóa “Using” trước bất kì từ khóa nào. Trình tự “Using” theo bảng chữ cái, trừ “Using System” luôn được them đầu tiên | Using System;  Using Entity…;  Using Math;  Namespace … |
| Thành viên Class | Các thành viên trong một “Class” được tổ chức theo trình tự:  -Nested classes, enums, delegates và events.  -Static, const và readonly fields.  -Fields và properties.  -Constructors và finalizers.  -Methods.  Within each group, elements should be in the following order:  -Public.  -Internal.  -Protected internal.  -Protected.  -Private.  Where possible, group interface implementations together. | Public class Father  {  //Nested class  Public class Son  {    }  //Enum  Public enum Wife  {  }  //Delegate  Public delegate void Date  {}  //Event  Public event Wedding  {}  // Trường Readonly  Private static readonly status;  // Trường và thuộc tính  private List<OrderItem> \_SoDo;  public List<OrderItem> SoDo;  //Contructor  Public Father(){};  //Phương thức  Public void Method(){}; |

## Quy tắc khoảng trắng

|  |  |
| --- | --- |
| Tên quy tắc | Ví dụ |
| Tối đa một câu lệnh trên mỗi dòng. | a++; // nên  b++;  a++; b++; // không nên |
| Tối đa một phép gán trên mỗi câu lệnh. | a = b; // nên  b = c;  a = b; b = c; // không nên |
| Khoảng trắng sau // và sau dấu phẩy, if else for while, … khoảng trắng giữa toán tử và mỗi toán hạng của tất cả các toán tử khác. | if (a < b) // nên  if(a<b) //không nên |
| Không ngắt dòng trước dấu mở ngoặc. | if (a < b) {  a = b;  } // nên  if (a < b)  {  a = b;  } // không nên |
| Không ngắt dòng giữa dấu ngoặc đóng } và else. | if (a < b) {  a = b;  } else {  b = c;  } |

## Cú pháp thân biểu thức

public string LayTenNguoiDung() {

// Phương thức thông thường

return \_tenNguoiDung;

}

public string LayTenNguoiDung => \_tenNguoiDung; // Thuộc tính với cú pháp thân biểu thức

## Cấu trúc và lớp

* Cấu trúc (structs) khác biệt rất lớn so với lớp (classes): Cấu trúc luôn được truyền và trả về bằng giá trị. Gán một giá trị cho một thành viên của một cấu trúc được trả về không làm thay đổi cấu trúc ban đầu.
* Hầu hết thời gian nên sử dụng lớp. Xem xét việc sử dụng cấu trúc khi kiểu có thể được xử lý giống như các kiểu giá trị khác.

## Biểu thức lambda và cách đặt tên:

Nếu một biểu thức lambda phức tạp (ví dụ: chứa nhiều hơn một vài câu lệnh, không bao gồm khai báo biến), hoặc được sử dụng lại ở nhiều vị trí khác nhau trong code, thì nên chuyển nó thành một phương thức được đặt tên rõ rang.

// Biểu thức lambda đơn giản

List<int> chan = soList.Where (so => so % 2 == 0); // Lọc số chẵn

// Biểu thức lambda phức tạp

List<string> tenTheoHoVaTenDem = nguoiDungList.Where (

nguoiDung => nguoiDung.Ho != null && nguoiDung.TenDem != null) .Select(nguoiDung => nguoiDung.Ho + " " + nguoiDung.TenDem); // Lọc người dùng có họ và tên đệm, sau đó tạo danh sách tên theo format "Họ Tên đệm"

//Nên biến thành một phương thức

private List<string> LayTenTheoHoVaTenDem (List<NguoiDung> nguoiDungList) {

return nguoiDungList.Where (

nguoiDung => nguoiDung.Ho != null && nguoiDung.TenDem != null) .Select (nguoiDung => nguoiDung.Ho + " " + nguoiDung.TenDem);

}

## Mảng và Danh sách

Ưu tiên List<> hơn arrays cho các biến public, thuộc tính, và kiểu trả về.

Ưu tiên List<> khi kích thước của container có thể thay đổi.

Ưu tiên arrays khi kích thước của container là cố định và biết trước tại thời điểm xây dựng.

Ưu tiên arrays cho các mảng đa chiều.

* + - * Lưu ý: array và List<> đều đại diện cho các container tuyến tính, liên tục.

Tương tự như arrays và std::vector trong C++, arrays có dung lượng cố định, trong khi List<> có thể thêm vào.

Trong một số trường hợp, arrays có hiệu suất cao hơn, nhưng nói chung List<> linh hoạt hơn.

## Vị trí thư mục và tệp

Nhất quán với dự án.

Ưu tiên một cấu trúc phẳng nếu có thể. (các tệp và thư mục trong một dự án mà tất cả các tệp đều được đặt trực tiếp trong cùng một thư mục gốc, không có các thư mục con bổ sung. Điều này có nghĩa là không có bất kỳ cấp thư mục con nào, tất cả các tệp đều nằm ở cùng một "mức" trong cấu trúc thư mục.)

## Đặt tên tham số

*// Không nên, những tham số này là gì?*

DecimalNumber product = **CalculateProduct**(values, 7, **false**, **null**);

*// Nên*

ProductOptions options = **new** **ProductOptions**();

options.PrecisionDecimals = 7;

options.UseCache = CacheUsage.DontUseCache;

DecimalNumber product = **CalculateProduct**(values, options, completionDelegate: **null**);

## Gọi hàm Delegate:

Sử dụng Invoke() và toán tử điều kiện null

Ví dụ: SomeDelegate?.Invoke()

Cách viết này giúp code rõ ràng hơn và xử lý an toàn với các delegate có thể null.

## Từ khóa var:

Từ khóa var trong C# cho phép khai báo biến mà không cần chỉ định rõ ràng kiểu dữ liệu của biến đó. Việc sử dụng var được khuyến khích khi nó giúp cải thiện tính dễ đọc của code.

**Trường hợp nên dùng var:**

**Kiểu dữ liệu rõ ràng:** Khi kiểu dữ liệu của biến dễ dàng đoán biết từ ngữ cảnh, sử dụng var giúp code gọn hơn. Ví dụ:

var apple = new Apple(); // Rõ ràng apple là kiểu Apple

var request = Factory.Create<HttpRequest>(); // Factory.Create trả về HttpRequest

**Biến tạm thời:** Sử dụng var cho các biến tạm thời, chỉ được truyền trực tiếp sang các phương thức khác. Ví dụ:

var item = GetItem(); // Kiểu của item phụ thuộc vào GetItem()

ProcessItem(item);

**Trường hợp không nên dùng var:**

**Kiểu dữ liệu cơ bản:** Không nên dùng var cho các kiểu dữ liệu cơ bản như int, bool, string vì kiểu dữ liệu đã rõ ràng. Ví dụ:

var success = true; // Kiểu bool đã rõ ràng, không cần var

**Kiểu số tự động suy luận phức tạp:** Nếu kiểu dữ liệu được suy luận từ các phép toán phức tạp, việc dùng var có thể gây khó hiểu. Ví dụ:

var number = 12 \* ReturnsFloat(); // Kiểu trả về của ReturnsFloat() không rõ rang

**Cần biết rõ kiểu dữ liệu:** Trong trường hợp người đọc code cần biết rõ kiểu dữ liệu của biến để hiểu logic, không nên dùng var. Ví dụ:

var listOfItems = GetList(); // Kiểu của danh sách trả về từ GetList() không rõ ràng

## Thuộc tính:

Trong lập trình C#, Attribute (Thuộc tính) được sử dụng để cung cấp thêm thông tin cho các thành phần code như field (trường), property (thuộc tính) hoặc method (phương thức). Vị trí đặt Attribute cũng ảnh hưởng đến tính dễ đọc và bảo trì code.

**Nguyên tắc:**

* Mỗi Attribute nên được đặt trên một dòng riêng, **ngay phía trên** thành phần code mà nó liên quan đến (field, property, method).
* Giữa Attribute và thành phần code được ngăn cách bởi một dấu xuống dòng (\n).
* Nếu có nhiều Attribute cùng áp dụng cho một thành phần, cần **ngăn cách** mỗi Attribute **bằng một dấu xuống dòng**.

**Lý do:**

* Việc này giúp dễ dàng thêm hoặc xóa Attribute trong tương lai.
* Mỗi Attribute riêng biệt và dễ dàng tìm kiếm trong code.

**Ví dụ:**public class ThongTinNguoiDung {

public string HoTen { get; set; }

public int Tuoi { get; set; }

}