Giáo trình: Xóa mù OOP

(Lập trình hướng đối tượng)

Ngôn ngữ lập trình: C++

Tác giả: Nguyên Nguyễn

Với sự hỗ trợ của AI

Mục lục

[A,Các thứ tạo nên class cơ bản 2](#_Toc21465)

[B) 4 tính chất của lập trình hướng đối tượng 6](#_Toc10865)

[Cú pháp để khai báo tính kế thừa: 8](#_Toc22211)

[Có 2 loại đa hình chính: 8](#_Toc1964)

[C) Các đơn vị kiến thức khác 11](#_Toc16587)

[Ví dụ: 11](#_Toc10639)

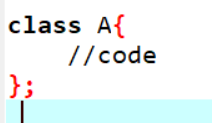
[D) Phần Main() 14](#_Toc3023)

[E) Bài tập tổng hợp 20](#_Toc17563)

[Lời kết: 22](#_Toc7167)

# A,Các thứ tạo nên class cơ bản

## I, Cấu trúc class cơ bản

****

## II, Các khoá thế hiện mức độ truy cập

### 1.Các mức truy cập

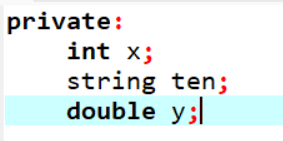
-**private:** Được ví như két sắt. Chỉ mình mới biết mật mã

-**public:** Cửa chính mở toang. Ai cũng có thể vào. Kể cả thằng trộm cũng có thể vào được.

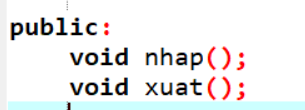
-**protected:** Cửa chính chỉ được mở cho con cháu họ hàng.

### 2, Các mức truy cập được sử dụng như thế nào

**-Private**: chủ yếu sẽ để các biến thành viên

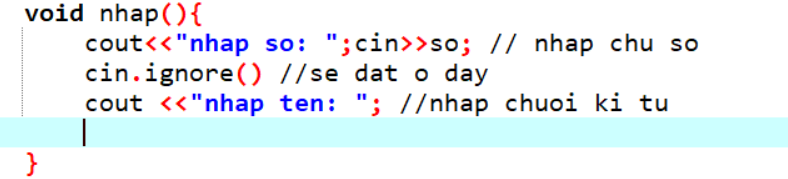


-**Public**: chủ yếu sẽ để các hàm xử lí.



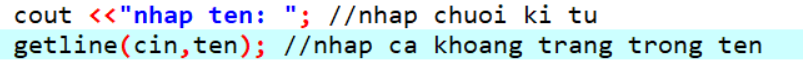
**Lưu ý**: Trong hàm nhap() sẽ có 2 hàm làm cho việc viết hàm nhap() trở nên trơn tru hơn.

- **cin.ignore()**: Dùng để xóa kí tự thừa trong bộ đệm



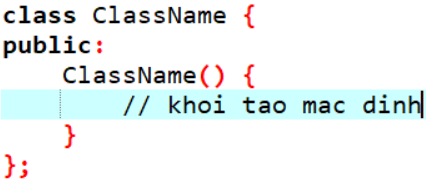
Ta sẽ đặt hàm cin.igonre() giữa kiểu dữ liệu int, double, float với string

- **getline():** dùng để nhập cả khoảng trắng.

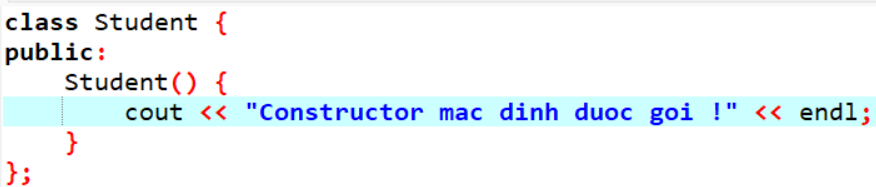
III, Constructor: hàm khởi tạo

### Constructor không tham số

Cú pháp:

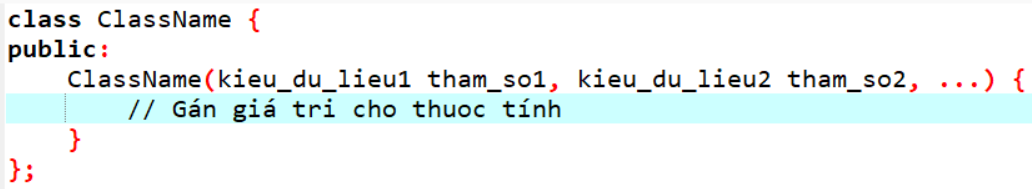


Ví dụ:

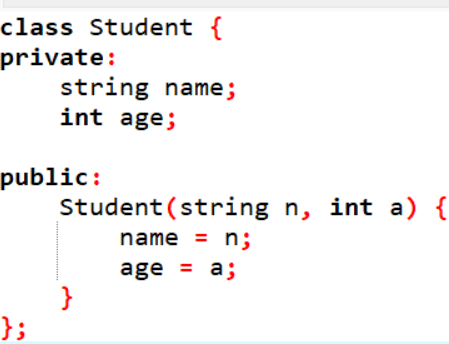


### Constructor có tham số

Cú pháp:



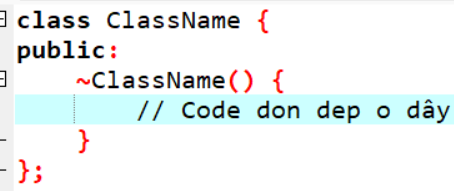
Ví dụ:



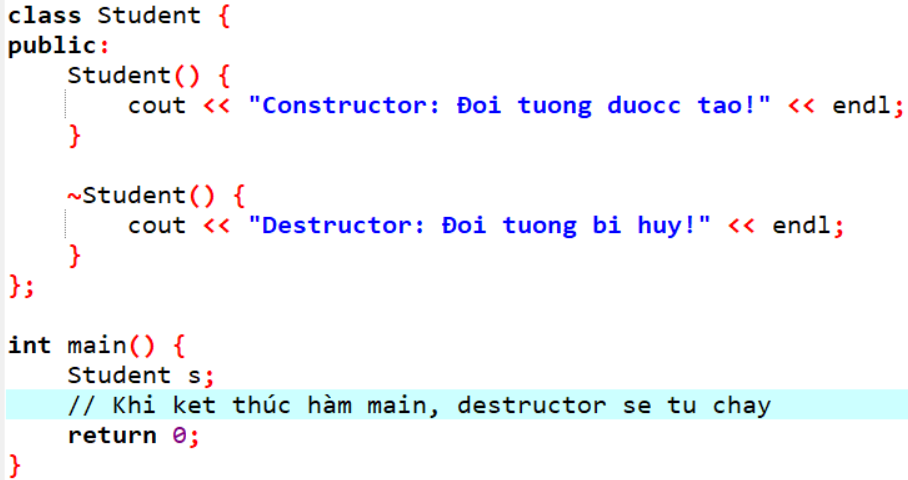
## IV) Destructor:

**Khái niệm: Destructor** là một hàm đặc biệt được gọi **tự động khi đối tượng bị hủy**, dùng để **giải phóng tài nguyên**, **dọn dẹp bộ nhớ**, hoặc **in thông báo thoát**.

Cú pháp:



Ví dụ



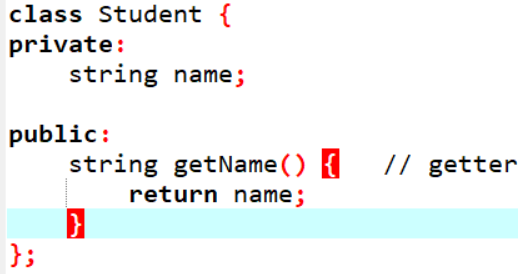
Chú ý: Nếu bạn có dùng con trỏ trong class (như cấp phát bằng new) thì destructor **cực kỳ quan trọng** để tránh **rò rỉ bộ nhớ**!

# B) 4 tính chất của lập trình hướng đối tượng

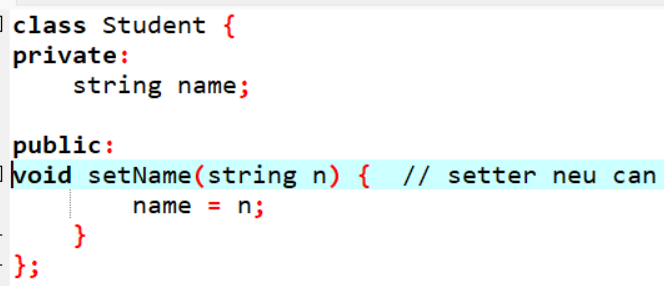
## 1, Tính đóng gói

Ví dụ đơn giản về tính đóng gói: Chúng ta có một ô vuông khép kín. Ở trong đó có 1 dấu chấm. Nếu chúng ta muốn lấy dấu chấm đó ra, chúng ta sẽ dùng hàm getter() để lấy dấu chấm đó là và dùng setter() để cập nhật giá trị.

Cú pháp hàm getter():



Cú pháp hàm setter():

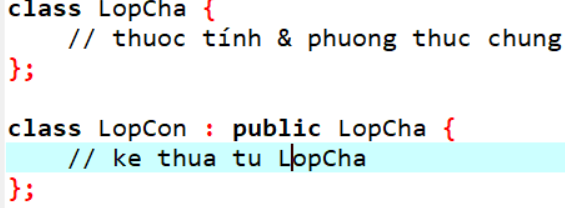


**Ghi chú:** Nếu bạn là người mới học lập trình hoặc đang học lại từ đầu, thì có thể tạm thời sử dụng **public** cho tất cả các thuộc tính để dễ quan sát và thử nghiệm. Tuy nhiên, khi đã hiểu rõ hơn về lập trình hướng đối tượng, bạn nên sử dụng **private** để đảm bảo an toàn cho dữ liệu – đây là nguyên tắc quan trọng trong **tính đóng gói (Encapsulation)**.

## 2,Tính kế thừa

Ví dụ về tính kế thừa: Từ tính đóng gói, ta sẽ vẽ thêm ô thứ 2. Sau đó ta sẽ lấy chấm tròn từ ô thứ nhất sang ô thứ 2 và sẽ tô thêm 1 chấm tròn nữa. Đó chính là Tính kế thừa.

**Cú pháp để khai báo tính kế thừa:**

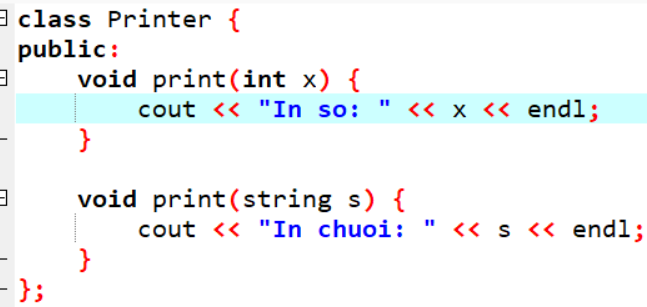
****

## 3, Tính đa hình.

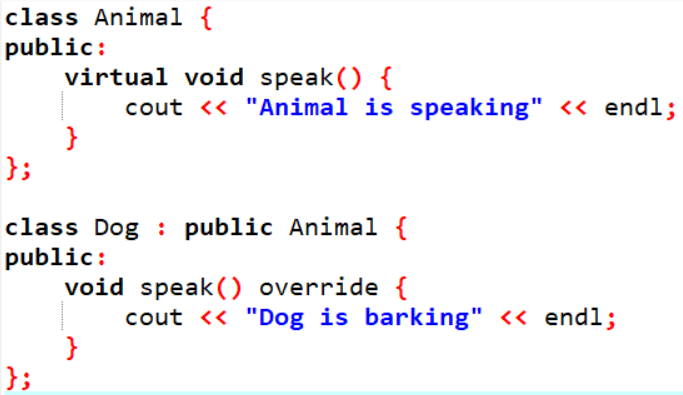
**Có 2 loại đa hình chính:**

**Nạp chồng (Overloading)**: cùng tên, khác tham số (compile time).  
**Ghi đè (Overriding)**: lớp con định nghĩa lại phương thức lớp cha (runtime).

Ví dụ về nạp chồng hàm:



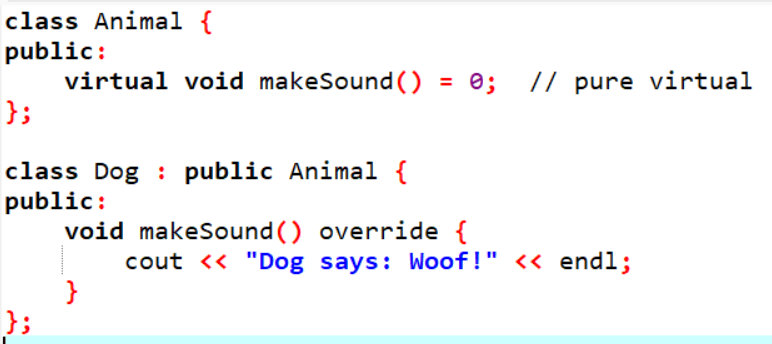
Ví dụ về ghi đè:



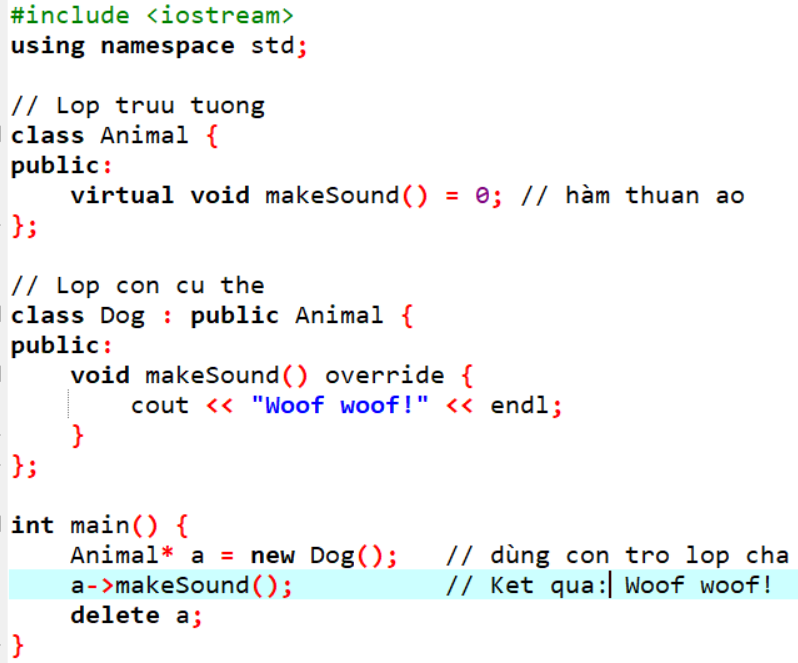
## 4, Tính trừu tượng:

**Tính trừu tượng**: Giúp **ẩn giấu logic phức tạp**, chỉ hiển thị phần cần thiết.

C++ dùng **pure virtual function.**



Ví dụ:



# C) Các đơn vị kiến thức khác

## I) Nạp chồng toán tử:

**Operator overloading** cho phép bạn định nghĩa lại **cách hoạt động của các toán tử** (+, -, ==, <<, v.v.) với các **kiểu dữ liệu do bạn tự định nghĩa (class)**.

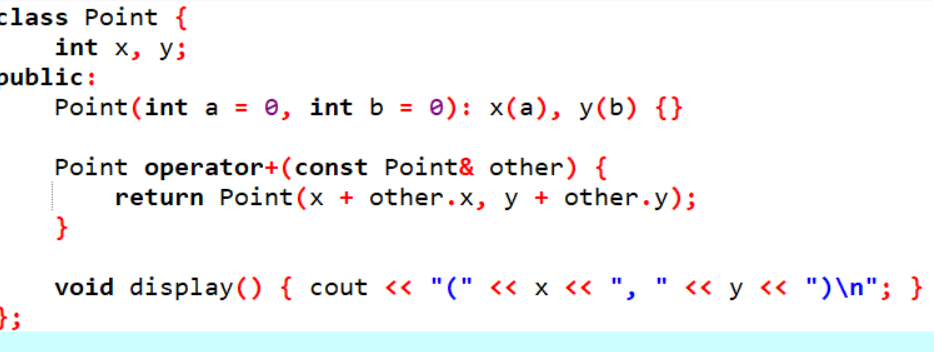
Ví dụ:

* Khi bạn viết a + b với hai số → máy hiểu ngay.
* Nhưng với a + b là **hai đối tượng** → bạn phải dạy máy **cách cộng chúng**, bằng cách **nạp chồng toán tử +**.

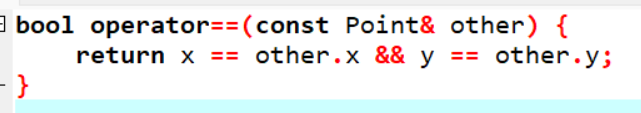
### 1. Toán tử + (hàm thành viên)

****

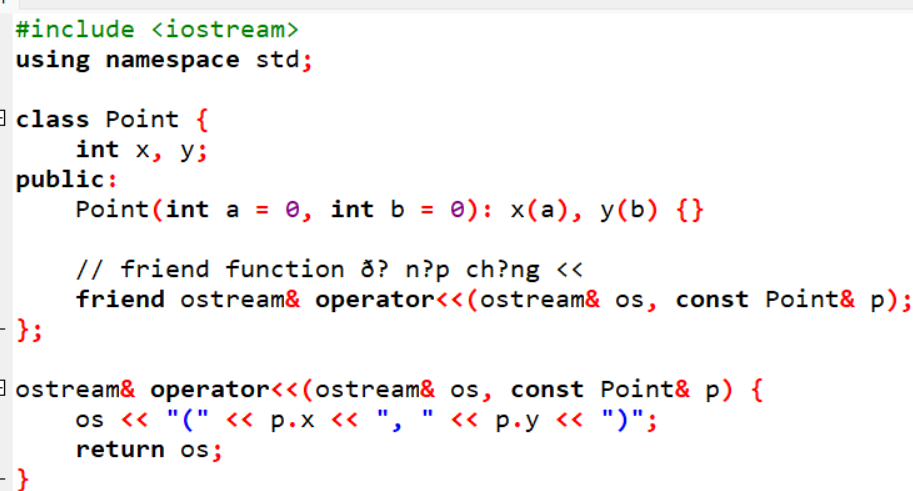
**Ví dụ:**

****

### 2. Toán tử so sánh



### 3. Toán tử >> và <<



## II) Hàm bạn , lớp bạn.

Cho phép một hàm hoặc class **truy cập vào thành phần private** của class khác. Thường dùng để nạp chồng toán tử <<, >>.

Cú pháp hàm bạn:



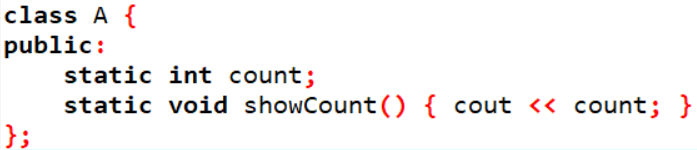
Cú pháp lớp bạn



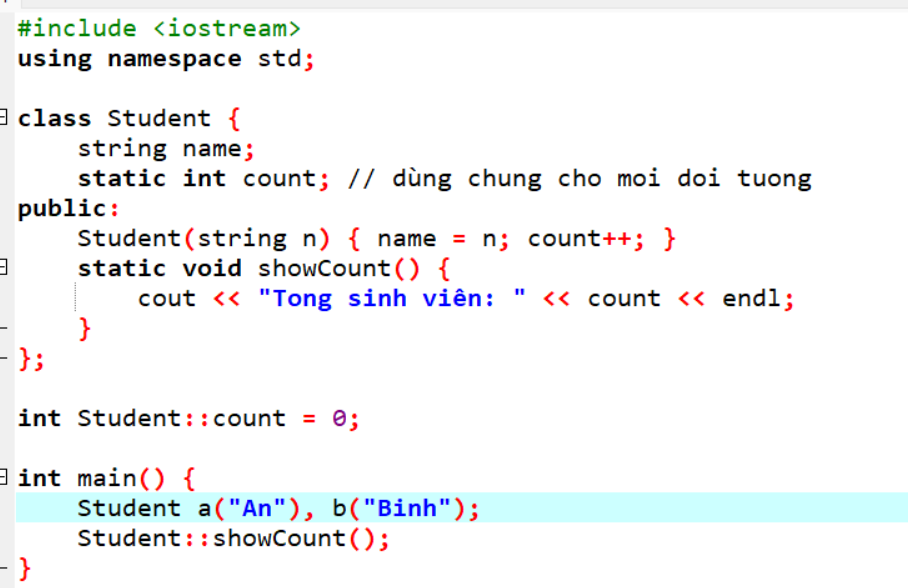
## III) Biến static

Dùng chung cho tất cả các đối tượng. Không phụ thuộc vào từng instance.

**Biến static**: dùng để đếm số đối tượng chẳng hạn.  
**Hàm static**: gọi trực tiếp qua class.



Ví dụ:

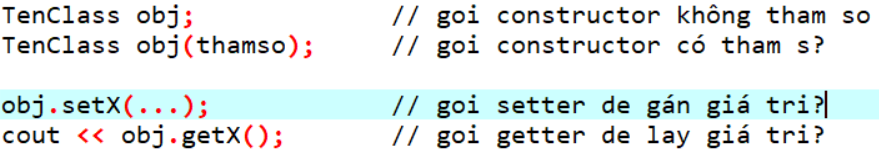


# D) Phần Main()

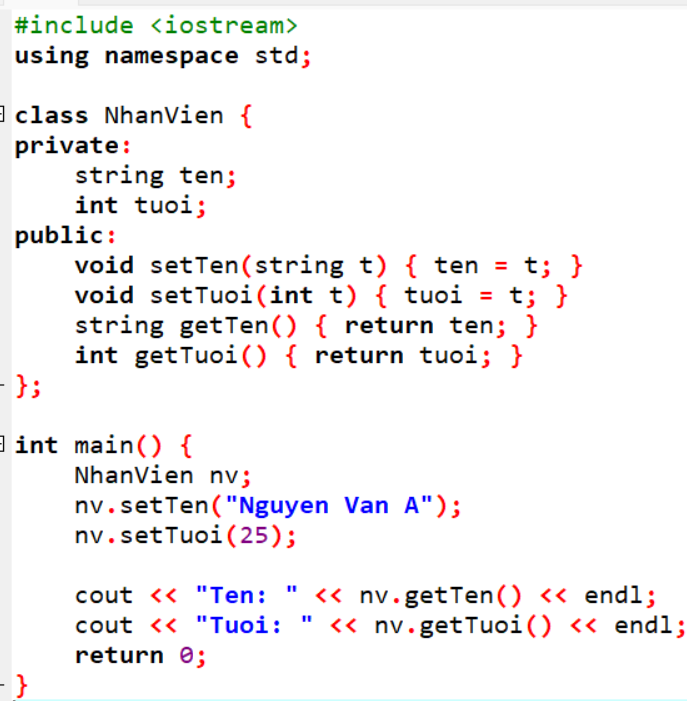
## I) Khai báo một đối tượng

### 1) Tính đóng gói

1. Cấu trúc



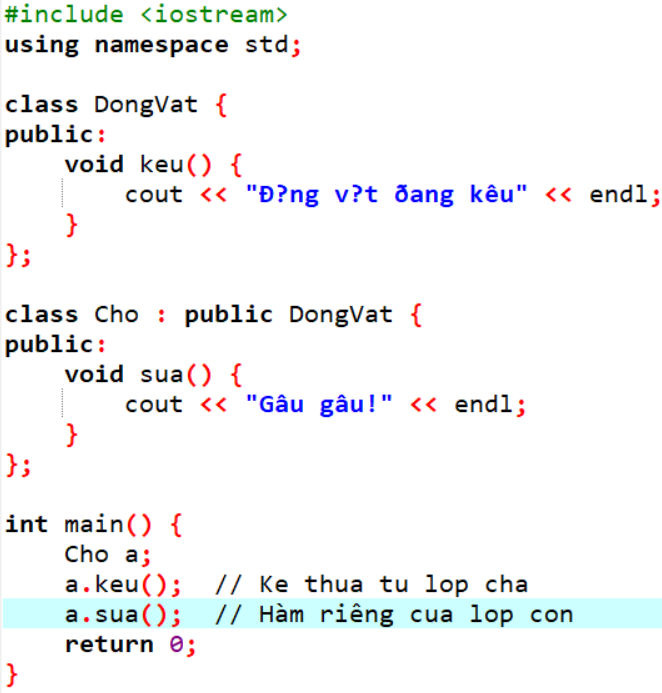
1. Ví dụ:



### 2) Tính kế thừa

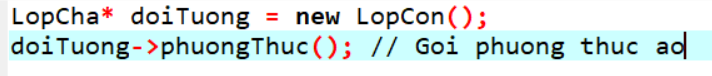
a)Cautruc

1. Ví dụ:

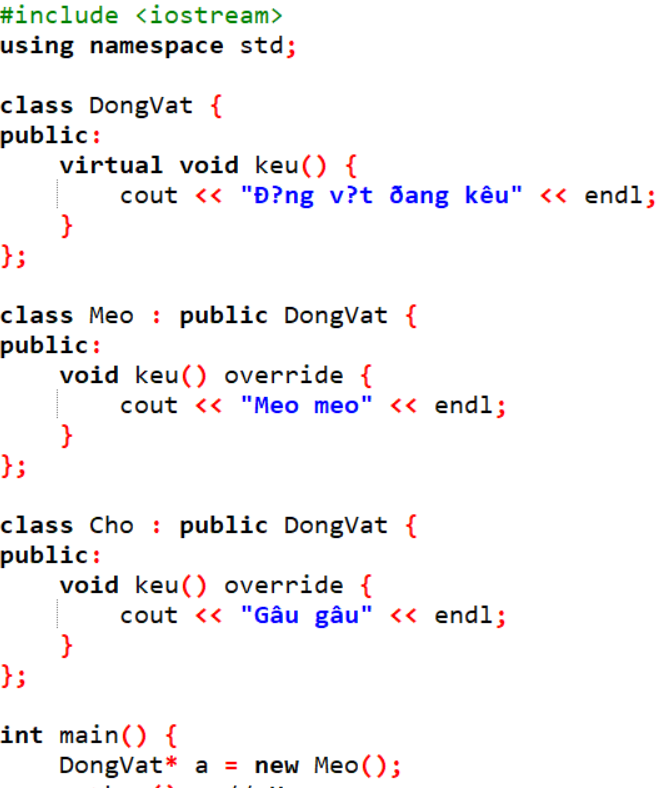


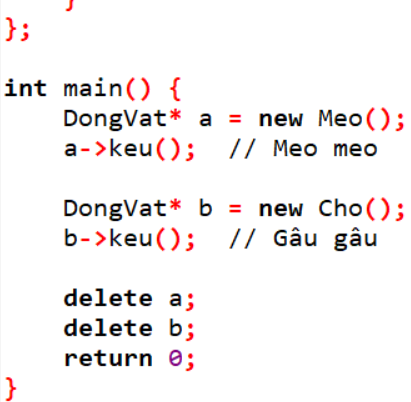
### 3) Tính trừu tượng:

a)Cấu trúc:

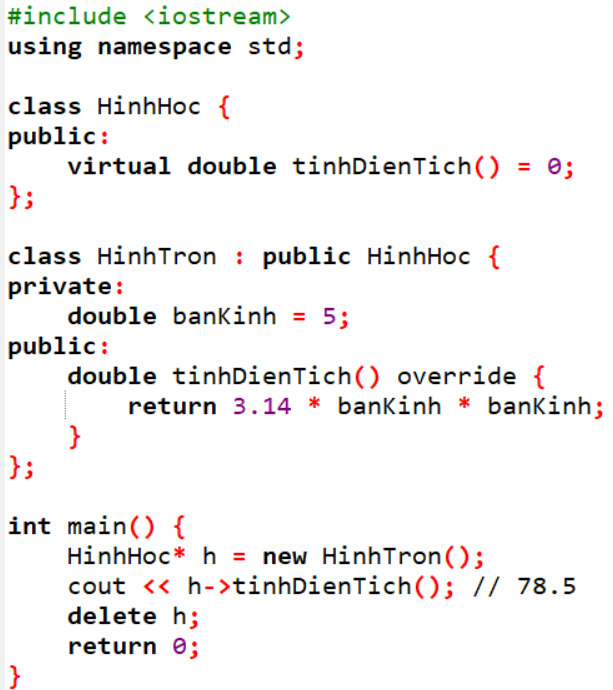


b)Ví dụ:





### 4) Tính trừu tượng

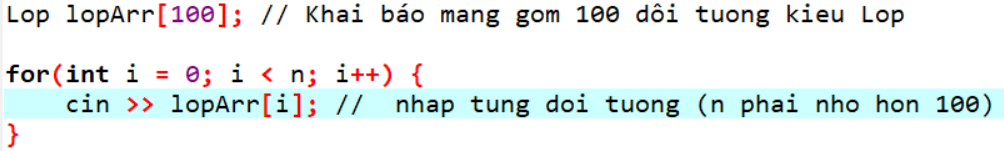


## II) Danh sách đối tượng

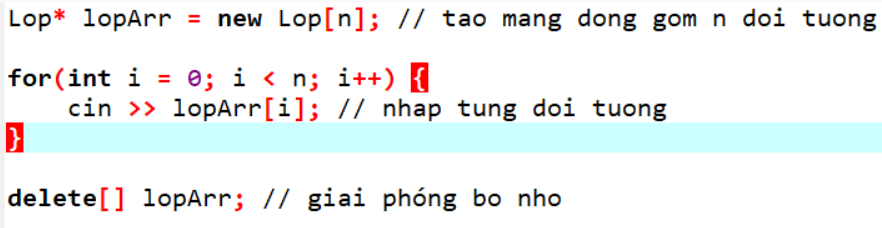
## 1)Khai báo:

Ta sẽ có 3 cách khai báo để có thể tạo danh sách các phần tử:

+Cách 1: Dùng mảng tĩnh

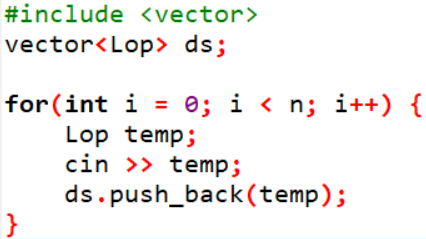


⚠ Cảnh báo: Giới hạn cố định, không thay đổi kích thước sau khi khai báo.

+Cách 2: Dùng mảng động: 

✅ Linh hoạt hơn mảng tĩnh. Tuy nhiên, cần **quản lý bộ nhớ thủ công**.

+Cách 3: Dùng vector

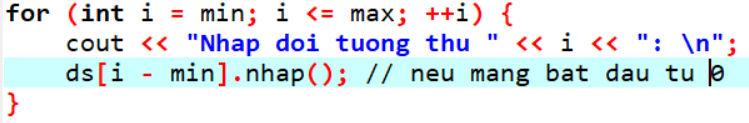


✅ Linh hoạt, tiện dụng, tự động quản lý bộ nhớ.

➕ Dễ dàng thêm/xoá phần tử với **.push\_back(), .erase()...**

2) for

1. Cấu trúc:



1. Lưu ý:

“Khung **for** từ min đến max giúp dễ điều chỉnh số lượng đối tượng cần nhập mà không bị giới hạn cứng. Đây là cách tổ chức linh hoạt, phù hợp với nhiều bài toán quản lý.”

# Bài tập tổng hợp

Viết chương trình quản lý sinh viên với các yêu cầu sau:

-Tạo lớp SinhVien gồm các thuộc tính:

-Mã sinh viên,họ tên, điểm toán, điểm văn, điểm anh.  
Các thuộc tính để **private.**

-Viết đầy đủ getter/setter, constructor, destructor  
-Tạo hàm tinhDiemTrungBinh() cho sinh viên.  
-Viết hàm in ra thông tin sinh viên.  
-Tạo lớp SinhVienCNTT kế thừa từ **SinhVien**, thêm điểm lập trình.  
-Ghi đè hàm tinhDiemTrungBinh() trong lớp **SinhVienCNTT.**

-Trong hàm main(), thực hiện:  
 Nhập danh sách sinh viên (tối thiểu 3 sinh viên).  
 In danh sách sinh viên ra màn hình.   
 Tìm sinh viên có điểm TB cao nhất.  
 Sắp xếp sinh viên theo điểm TB giảm dần.

**Lời kết:**

Giáo trình "Xóa mù OOP" được biên soạn với tâm huyết nhằm giúp các bạn dễ dàng tiếp cận lập trình hướng đối tượng bằng C++ và Java, theo cách đơn giản – dễ hiểu – dễ áp dụng.

Một phần nội dung trong giáo trình có sự hỗ trợ của trí tuệ nhân tạo (AI) để đảm bảo tính hệ thống, khoa học và rõ ràng. Tuy nhiên, toàn bộ cấu trúc, phong cách, ví dụ minh họa, bài tập... đều được thiết kế theo cách riêng – gần gũi với người học Việt Nam, đặc biệt là các bạn mới bắt đầu.

Hy vọng giáo trình này sẽ là người bạn đồng hành hữu ích trên hành trình học lập trình của bạn.  
 Đừng quên rằng, học lập trình không khó – quan trọng là học đúng cách và đúng lộ trình.

Chúc bạn học tốt và thành công!