**DIAGRAM & CON TRỎ**

1. **DIAGRAM**
2. **Relationship between classes**
3. **Inheritance**

Có tên gọi khác:

* Quan hệ tổng quát hóa
* Quan hệ khái quát hóa
* Quan hệ kế thừa: Đối tượng cụ thể (concrete) sẽ kế thừa các thuộc tính và phương thức của đối tượng tổng quát (general).

1. Tính chất

* Mối quan hệ giữa các lớp trong đó một lớp chia

sẻ thuộc tính và/hoặc phương thức với một hoặc nhiều

lớp khác

* Xác định sự phân cấp về mức độ trừu tượng hóa

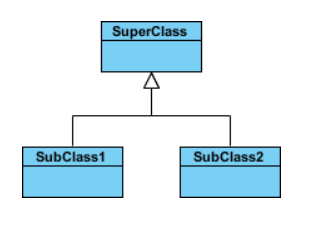
trong đó lớp con kế thừa từ một hoặc nhiều lớp

cha

* + Đơn kế thừa (Single inheritance)
  + Đa kế thừa (Multiple inheritance)
* Là mối quan hệ “is a”
* Tên lớp trừu tượng được in nghiêng.

1. Biểu diễn

Inheritance được biểu diễn bằng một đường thẳng được kết nối với một đầu mũi tên rỗng hướng về phía lớp cha.



1. **Dependency**

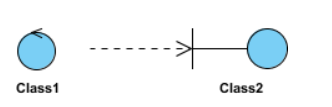
Một đối tượng của một class1 có thể sử dụng một đối tượng của một class2 khác trong một phương thức của class1. Nếu đối tượng (class1) không được lưu trữ trong bất kỳ trường nào (của class2), thì điều này được mô hình hóa như một mối quan hệ phụ thuộc. Ta nói class1 phụ thuộc vào class2.

1. Tính chất

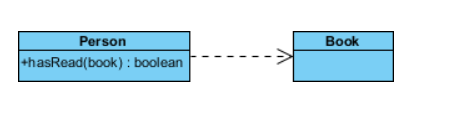
* Là một kiểu liên kết đặc biệt
* Tồn tại giữa hai lớp nếu những thay đổi đối với định nghĩa của một lớp có thể gây ra những thay đổi cho lớp kia (nhưng không phải ngược lại).
* Class1 phụ thuộc vào Class2

1. Biểu diễn

* Hình dưới đây cho thấy một ví dụ về sự phụ thuộc. Mối quan hệ được hiển thị dưới dạng đường đứt nét với một mũi tên mở.



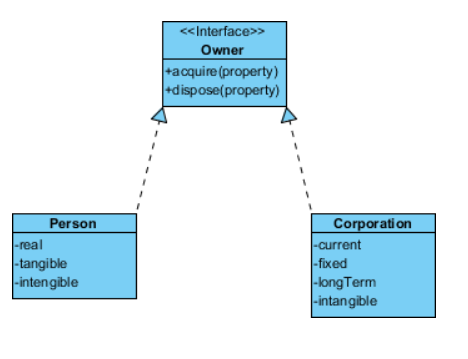
* Hình dưới đây cho thấy một ví dụ khác về sự phụ thuộc. Lớp Person có thể có phương thức hasRead với tham số Book trả về true nếu người đó đã đọc sách (có thể bằng cách kiểm tra một số cơ sở dữ liệu).



1. **Realization**

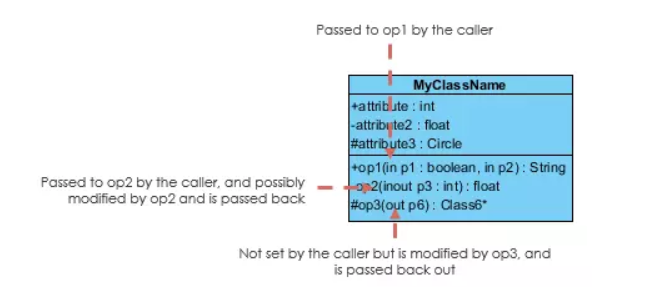
Realization là một mối quan hệ giữa lớp kế hoạch chi tiết và đối tượng chứa các chi tiết mức độ thực thi tương ứng của nó. Đối tượng này được cho là nhận ra lớp kế hoạch chi tiết. Nói cách khác, bạn có thể hiểu đây là mối quan hệ giữa interface và implementting.

Ví dụ: <<Interface>> Owner có thể chỉ định các phương pháp acquire(property) và dispose(property). Các lớp Person và Corporation cần triển khai các phương thức này, có thể theo những cách rất khác nhau.



1. **Parameter Directionality (Định hướng tham số)**

Mỗi tham số trong một hoạt động (phương thức) có thể được ký hiệu là in, inout hoặc out trong đó chỉ định hướng của nó đối với người gọi. Hướng này được hiển thị trước tên tham số.



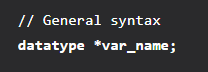
1. **In:** Một biến được truyền trong chế độ này có tính chất chỉ đọc. Điều này có nghĩa là, giá trị không thể thay đổi và phạm vi của nó bị hạn chế trong hàm. Hàm nhận một giá trị từ đối số này khi hàm được gọi.
2. **Out:** Trong chế độ này, một biến chỉ được ghi và có thể được chuyển trở lại chương trình đang gọi. Nó không thể được đọc bên trong hàm và cần được gán một giá trị.
3. **Inout:** Hàm có thể đọc giá trị của các biến và cũng có thể thay đổi nó để chuyển nó đến hàm đang gọi.

* Nghĩa là:tham số có thể đọc giá trị và có thể được ghi đè.

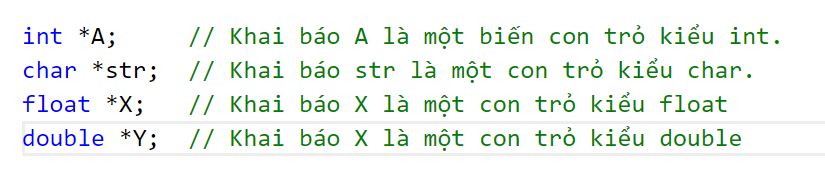
1. **SỬ DỤNG CON TRỎ**
2. **Con trỏ**
3. **Định nghĩa:** Con trỏ là một biến mà giá trị của nó là địa chỉ của một biến khác.
4. **Tính chất**
   * Một biến con trỏ được phép trỏ đến bất kỳ đối tượng nào có cùng kiểu với nó.
   * Một biến con trỏ được phép biến đổi gián tiếp giá trị của biến mà con trỏ trỏ đến.
   * Một biến con trỏ có thể trỏ đến bất kỳ một miền nhớ nào, thiết lập giá trị của miền nhớ đó, thay đổi nội dung của miền nhớ.
   * Nếu A là một biến thì địa chỉ của A trong bộ nhớ là &A .
5. **Khai báo**

Kiểu con trỏ phải được định nghĩa trên một kiểu cơ sở đã được định nghĩa trước đó.

* *Syntax*

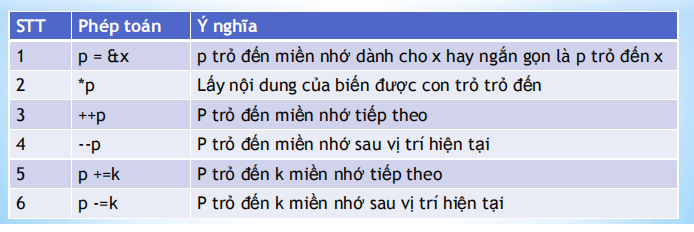
******

* *Ví dụ*

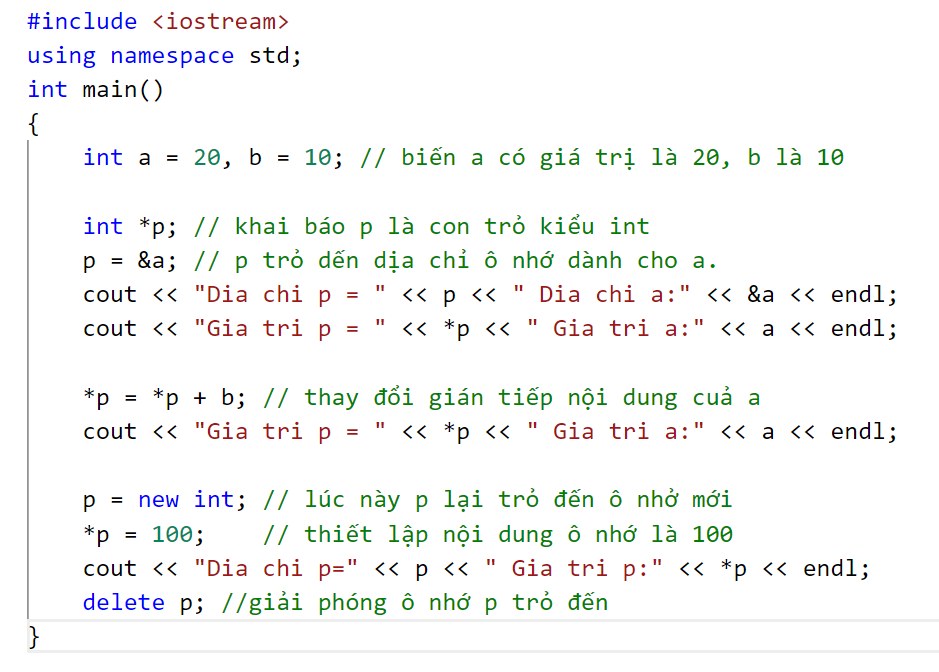
**

1. **Các phép toán trên con trỏ**

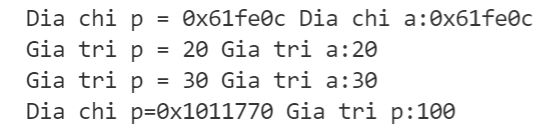
Giả sử p là một con trỏ và x là biến có cùng kiểu. Khi đó:



* Ví dụ: Thay đổi nội dung của biến thông qua con trỏ.



* Output:



1. **Con trỏ và mảng**
2. **Khái quát**

* Một mảng được xem là một hằng con trỏ. Tên của mảng là địa chỉ của mảng trong bộ nhớ.
* Ví dụ ta có mảng int A[10]. Hệ thống cấp phát một miền nhớ là

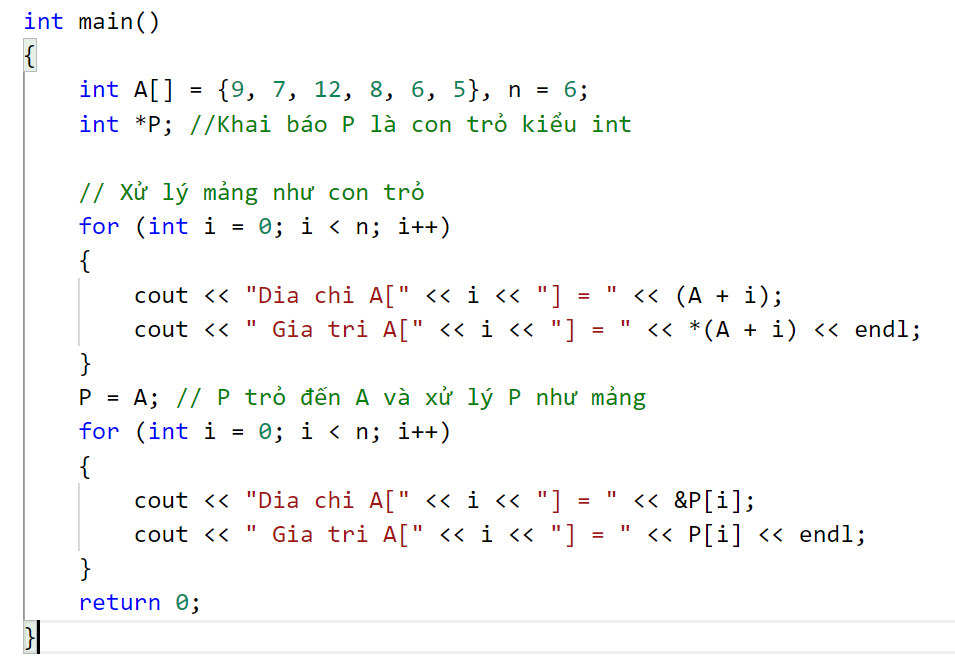
10\*sizeof(int). Địa chỉ phần tử đầu tiên &A[0] được qui ước là tên mảng A. Địa chỉ phần tử thứ i là (A+i) = &A[i]. Giá trị của phần tử thứ i là \*(A+i) = A[i].

* Một con trỏ p được phép trỏ đến một mảng A theo chỉ thị p = A. Sau đó p được thực hiện các thao tác như một mảng: p[0] = 5, p[3] = 7. Nhưng mảng A thì không được phép trỏ đi đâu cả vì nó là một hằng con trỏ.
* Khi sử dụng một mảng trong lập trình, ta không biết khai báo số phần tử của

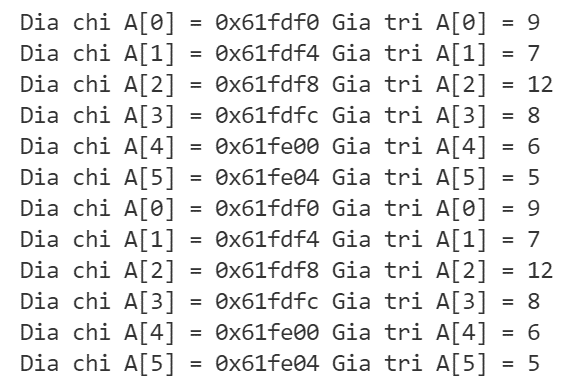
mảng là bao nhiêu cho đủ. Nếu số lượng phần tử nhỏ thì sợ thiếu không gian

nhớ, nếu số lượng phần tử lớn lại gây lãng phí bộ nhớ. Trong tình huống này ta nên sử dụng con trỏ thay thế cho mảng thông qua hai chỉ thị:

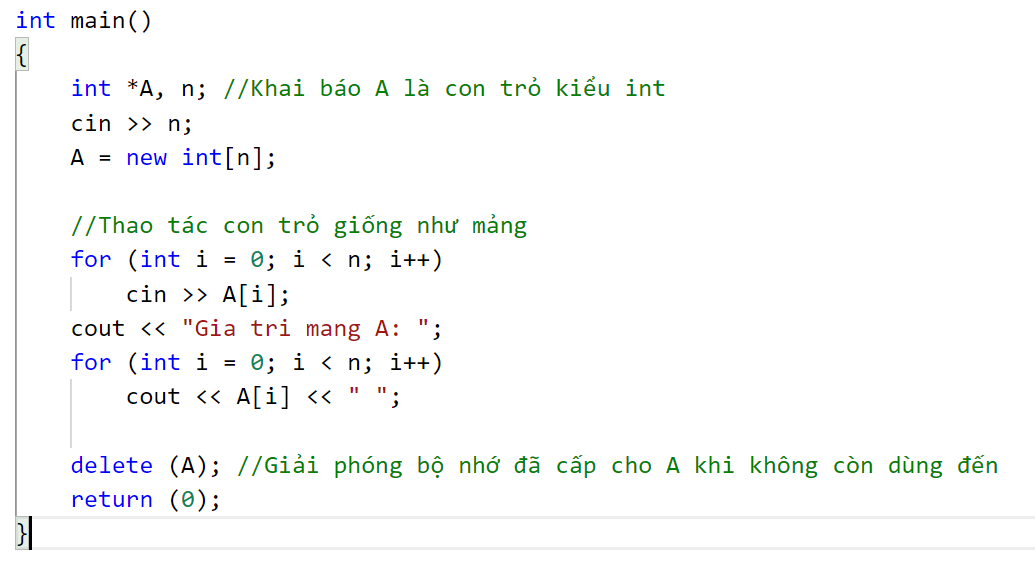
* Chỉ thị new: cấp phát miền nhớ cho con trỏ.
* Chỉ thị delete: giải phóng miền nhớ cho con trỏ
* Ví dụ:
* Sử dụng mảng giống như con trỏ và sử dụng con trỏ giống như mảng.

****

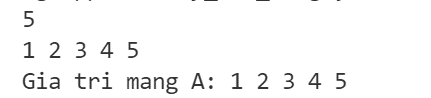
* Output:



* Thay thế mảng bằng con trỏ.



* Output:

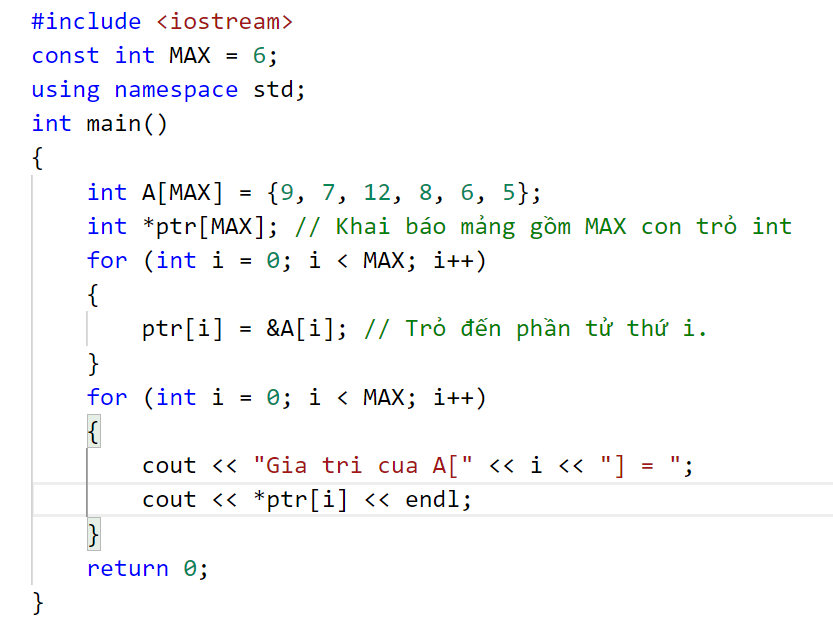


1. **Mảng các con trỏ**

* **Định nghĩa:** Mảng các con trỏ là một mảng mà mỗi phần tử của nó là một con trỏ.
* **Khai báo:**

****

* **Ví dụ:** Truy cập mảng 1 chiều bằng con trỏ

****

* **Output:**

