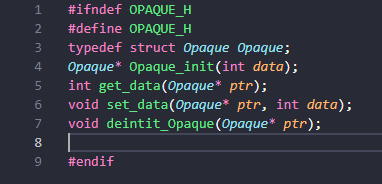
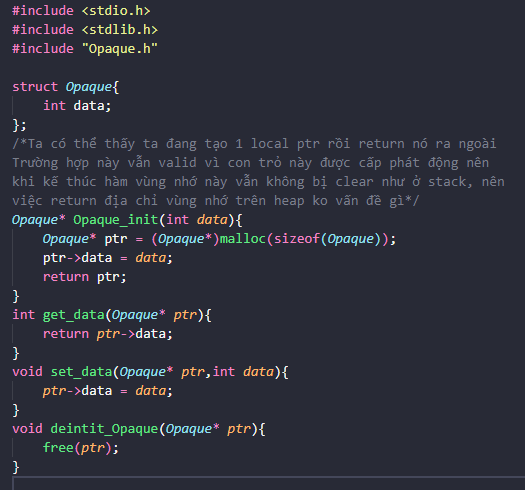
* **Object pattern:** 
  + The fastest way to make your code cleanly structured and easy to understand
  + Là một kĩ thuật để mô phỏng các khái niệm của lập trình hướng đối tượng.
  + Sử dụng **struct** để mô phỏng các đối tượng và **function pointer** để mô phỏng các method của đối tượng
* 
* Tư duy theo kiểu C++ thì ở trong struct ta sẽ có data như class, còn thay vì method như trong C++ ta sử dụng function pointer để trỏ đến các hàm như là method tương ứng.
* Chú ý cách sử dụng function pointer khi gọi method từ struct.

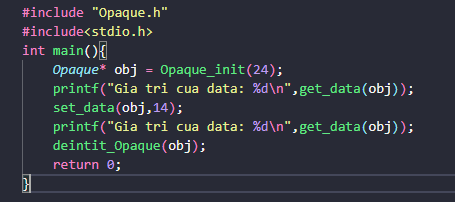
**Opaque Pattern**

* Mục tiêu: Đảm bảo tính đóng gói (encapsulation) trong ngôn ngữ C – nơi ko hỗ trợ OOP.
* Cách thức hoạt động:
  + Ẩn cấu trúc dữ liệu (struct): Opaque pattern sử dụng 1 struct nhưng không tiết lộ chi tiết nội bộ của nó. Thay vào đó chỉ sử dụng 1 con trỏ tới struct này và tất cả các method sẽ thông qua interface ( ví dụ là các hàm)
  + Cung cấp các hàm API để người dùng tương tác với struct đó
* Hiểu đơn giản thì ta có 1 struct, các hoạt động liên quan đến struct này sẽ phải thông qua các hàm API và không thể thay đổi hay truy cập trực tiếp struct từ bên ngoài.
* File **Opaque.h**

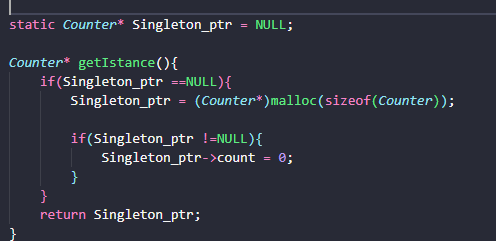


* File **Opaque.c**
* 

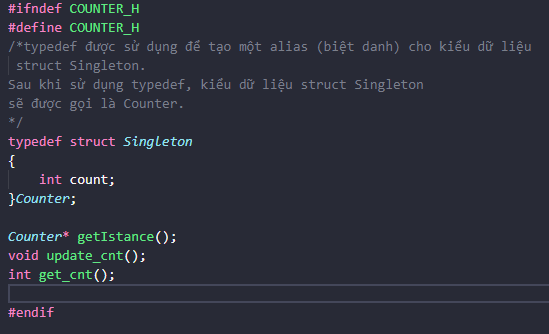
-File **main.c**

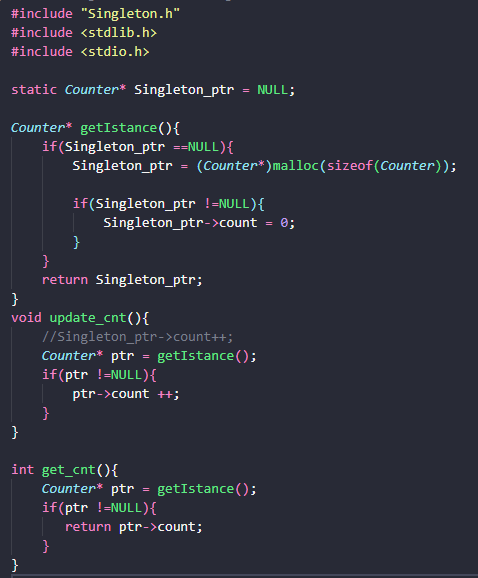


**Singleton Pattern**

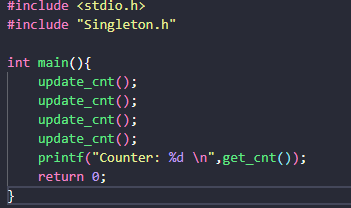
* Mục đích: Đảm bảo 1 struct chỉ có 1 instance ( thể hiện duy nhất) trong toàn bộ chương trình.
* Khi nào sử dụng Singleton Pattern?
  + Khi bạn cần đảm bảo rằng một lớp có duy nhất một instance trong toàn bộ chương trình. Ví dụ, nếu bạn có một đối tượng quản lý kết nối cơ sở dữ liệu hoặc một đối tượng ghi log, bạn chỉ muốn có một instance duy nhất của nó trong suốt vòng đời của ứng dụng.
  + Khi bạn cần quản lý tài nguyên hệ thống, như kết nối cơ sở dữ liệu hoặc tài nguyên chia sẻ, mà không có nhiều đối tượng cùng truy cập.
* Cách thức hoạt động:
  + Tạo 1 con trỏ **static global** kiểu struct đó để giữ instance duy nhất của struct
  + Phương thức GetInstance() sẽ kiểm tra xem con instance đã tồn tại chưa, nếu chưa thì cấp phát mới rồi return con trỏ đó.
  + 

Singleton.h



Singleton.c

Main.c



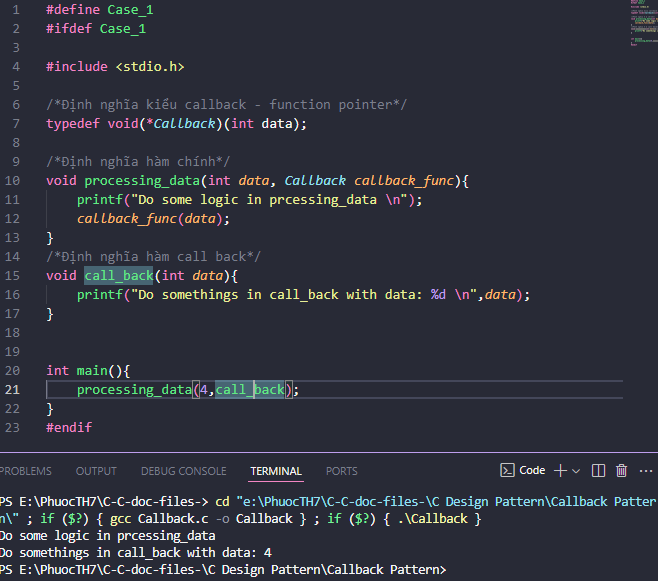
**Factory Pattern**

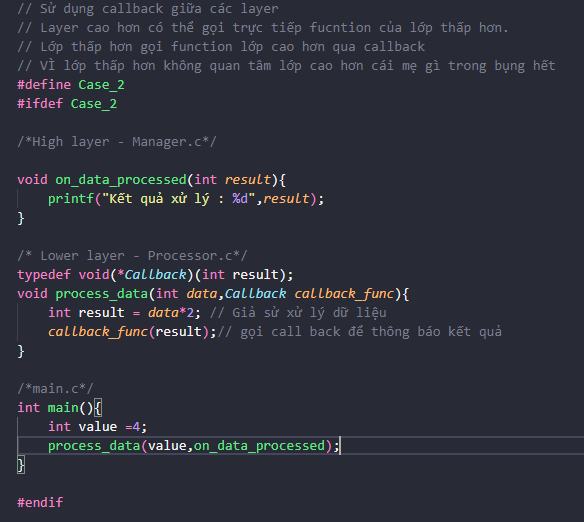
* Nói đơn giản là một pattern để tạo các đối tượng mới.
* Tạo ra 1 factory, mà khi ta muốn tạo đối tượng mới ta chỉ cần thông qua factory này, không cần quan tâm quá trình tạo đối tượng thực tế chi tiết xảy ra những gì.

**Structural Pattern:**

* Giải quyết các bài toán kết hợp các đối tượng và lớp -> Tạo thành 1 cấu trúc hợp lý và hiệu quả.

1. Call back Pattern:

* Nói một cách đơn giản thì Callback pattern là 1 mẫu thiết kế cho phép bạn truyền hàm A vào hàm B (dưới dạng tham số là function pointer), sau đó hàm B có thể gọi lại (callback) hàm A khi có điều kiện gì đó xảy ra.
* Lý do dùng call back:
  + Tăng tính linh hoạt: Ta có thể truyền nhiều hàm khác nhau vào 1 hàm và hàm đó sẽ thực thi các hành vi tùy chỉnh mà bạn muốn.
  + Một hàm có thể gọi lại hàm khác mà không cần biết chính xác nội dung của nó, chỉ cần hàm đó đúng kiểu và thực hiện chức năng mình mong muốn.
* Cơ chế hoạt động:
  + Khai báo và định nghĩa hàm callback: Tạo 1 function pointer để lưu địa chỉ hàm callback.
  + Truyền hàm callback vào hàm khác: Hàm A nhận hàm B (callback) như 1 tham số, sau khi hoàn thành công việc hàm A sẽ gọi hàm B được truyền vào.
* Ta xem các ví dụ sau:
  + Đây là ví dụ cơ bản về call back. Tất nhiên thì sử dụng call back trong cùng 1 file không có nhiều ý nghĩa. Ta sẽ tìm hiểu vì sao lại sử dụng call back sau.
* 



* Một ứng dụng phổ biến của callback pattern là việc layer thấp hơn gọi layer cao hơn.
* Hiểu một cách đơn giản thì trong dự án, layer cao hơn có thể gọi trực tiếp function của layer thấp hơn, nhưng ngược lại thì không được phép. Vì vậy ra phải sử dụng callback để hàm của layer thấp có thể gọi layer cao.
* Vậy tại sao lại không gọi trực tiếp mà lại phải dụng call back?
  + Nếu sử dụng gọi hàm trực tiếp, ta phải biết chính xác hàm nào sẽ được gọi và hoàn toàn kiểm soát việc gọi hàm đó. Điều này là không nên vì đã quy định là layer thấy hơn rồi mà còn biết hết về lớp trên như thế là không ổn.
  + Khi gọi qua callback, ta không trực tiếp gọi hàm mà ta truyền 1 con trỏ hàm cho hàm ở layer thấp, Ta không cần biết con trỏ này trỏ đến hàm nào mà chỉ quan tâm đúng kiểu là được. Nói đơn giản lớp thấp chỉ yêu cầu lớp cao thực hiện 1 hành động mà k quan tâm hành động đó chính xác là gì.