Anh nhớ em chẳng hạn !

- biến là 1 vùng trong bộ nhớ ram, dùng để lưu trữ thông rin, có thể gán thông tin cho 1 biến hoặc lấy thông tin ra sư dụng

- lập trình hướng đối tượng gồm 4 tính chất:

+ tính đóng rồi : là cơ chế ràng buộc dữ liệu của 1 đối tượng, tránh bị trách động bên ngoaif.

+ tính kế thừa : xây dựng các lớp mới thông qua lớp đã có, lớp con gọi là lớp dẫn xuất, lớp cha là lớp cơ sở.

+ tính đa hình: có nhiều tính chất từ 1 tính chất gốc

- struct chỉ có các thuộc tính, còn class thì có thêm các phương thức!

**- struct phân vùng bộ nhớ của các biến nằm rời rạc với nhau, còn union thì phân vùng bộ nhớ của các biến có thể nằm lên nhau. Ví dụ như struct float a(4 byte), int b(2 byte) và char c(1byte) thì cần 7 byte bộ nhớ, còn với union thì chỉ cần 4 byte của float là đủ, trong 4 byte đó chứ 2 byte của int b, trong b chứa 1 byte của char c !**

**=> do đó giá trị của các biến trong struct thì không ảnh hưởng nhau còn union thì bị ảnh hưởng nhau**

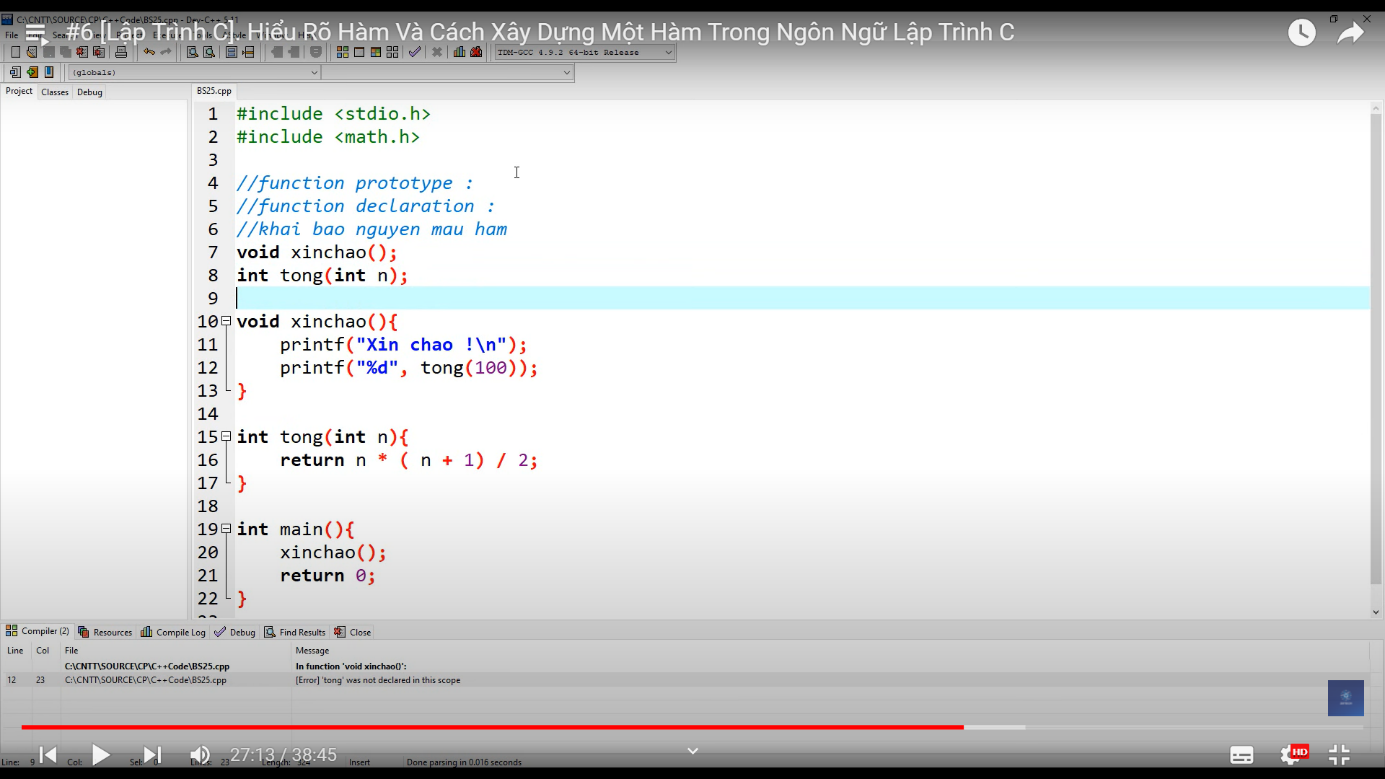
**=> sinh ra union để sử dụng trong trường hợp bài toán sử dụng 1 loại trong các biến vì tiết kiệm bộ nhớ.**

- vòng lặp while dùng trong trường hợp tính toán với các chữ số của số nhập vào, muốn lấy từng chữ số của số nhập vào ta chia lấy dư cho 10, còn lại chia lấy nguyên cho 10 thì sẽ đếm được số lượng chữ số.

- còn vòng lặp do..while là dùng trong trường hợp yêu cầu khi nhập sai thì nhập lại cho đúng.

*- parameter là tham số, chính là cái ở trong hàm, còn argument là đối số lại là cái mình truyền vào cho hàm tương ứng với đố số => KHI GỌI HÀM CHỈ ĐƠN THUẦN LÀ PHÉP GÁN GIÁ TRỊ CỦA ĐỐI SỐ CHO THAM SỐ VÀ THỰC HIỆN TRÊN THAM SỐ !!! ( ví dụ như ở trong hàm tăng đơn vị có biến n, xuống hàm main ta truyền biến m vào cho n, thì kết quả trả về là kết quả của n còn biến m vẫn giữ nguyên ! => để thay đổi giá trị của m trong trường hợp này thì ta phải dùng* ***CON TRỎ****)*

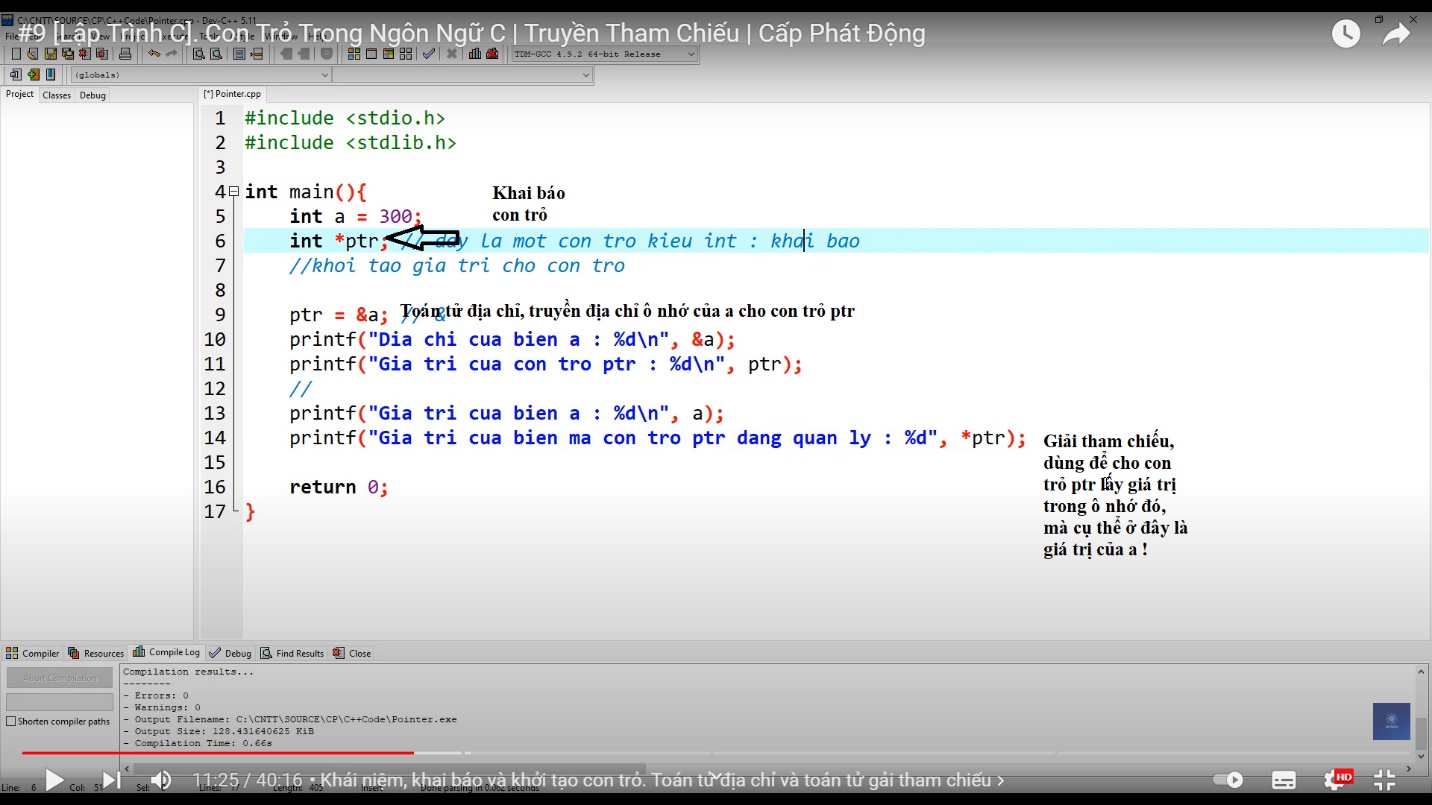
- các hàm có thể gọi lẫn nhau, nhưng mà hàm 1 được gọi trong hàm 2 thì hàm 1 phải được xây dựng trước hàm 2. Để tránh trường hợp này phải xây dựng hàm nguyên mẫu ( funtion declaration)

 Hình ảnh về xây dựng nguyên mẫu hàm để không bị lỗi

- khi xây dựng hàm để xác định tính chất đúng sai thì nên trả về giá trị 1 hoặc 0 để có thể tái sử dụng được nhiều lần, không cần cụ thể quá ở trong hàm.

- Nếu truyển mảng cho hàm thì lúc này mảng sẽ nhận giá trị như thực hiện khi ở trên hàm.

- CON TRỎ: dùng để gán giá trị của ô chứa biến đó ( địa chỉ của ô nhớ)



- thông thường chúng ta sử dụng hàm thì trong hàm main chúng ta sẽ có câu lệnh truyền giá trị cho hàm ( không làm thay đổi giá trị của đối số), nhưng sử dụng con trỏ để truyền tham chiếu thì sẽ thay đổi giá trị của đối số luôn. Nói cách khác khi tham số của hàm là con trỏ thì con trỏ này sẽ lấy luôn ô nhớ của đối số trong hàm main, có thể thay đổi luôn giá trị của đối số này !



- ứng dụng thường gặp của con trỏ là dùng để swap giá trị của 2 số !

- mảng thực chất là 1 hằng con trỏ, mảng thì là tập hợp của nhiều ô nhớ, vì vậy địa chỉ của mảng là địa chỉ của ô nhớ đầu tiên !

- có thể gán con trỏ cho 1 mảng: int \*ptr = a;

- lệnh getchar(); dùng để đọc 1 kí tự từ bàn phím: thường dùng để khi nhấn enter sẽ được tiêu biến khi chạy code.

MCU

-Mcu là viết tắt của microcontroller unit, là vi điều khiển, Vi điều khiển là một máy tính được tích hợp trên một chip, nó thường được sử dụng để điều khiển các thiết bị điện tử. Vi điều khiển, thực chất, là một hệ thống bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp (khác với các bộ vi xử lý đa năng dùng trong máy tính) kết hợp với các khối ngoại vi như bộ nhớ, các module vào/ra, các module biến đổi số sang tương tự và tương tự sang số,... Vi điều khiển thường được sử dụng để xây dựng các hệ thống nhúng. Nó cũng được sử dụng trong các thiết bị điện, điện tử như máy giặt, lò vi sóng, điện thoại, đầu đọc DVD, thiết bị đa phương tiện hay dây chuyền sản xuất tự động,...

Hầu hết các vi điều có 4 bốn thành phần cần thiết của một hệ thống nhúng.

Những thành phần này là :

* lõi CPU,
* bộ nhớ chương trình (thông thường là ROM hoặc bộ nhớ flash),
* bộ nhớ dữ liệu (RAM),
* một hoặc vài bộ định thời và các cổng vào/ra để giao tiếp với các thiết bị ngoại vi và các môi trường bên ngoài –

tất cả các khối này được thiết kế trong một vi mạch tích hợp. Vi điều khiển khác với các bộ vi xử lý đa năng ở chỗ là nó có thể hoạt động chỉ với vài vi mạch hỗ trợ bên ngoài.

**UART**

* **Giao tiếp uart (Universal Asynchronous Receiver – Transmitter)** là giao tiếp truyền nhận không đồng bộ, không có chân SCK và chân CS như là giao tiếp như SPI cho nên chỉ có 2 chủ thể giao tiếp với nhau. Sử dụng chân Tx và chân Rx. 2 chủ thể tương đương nhau. Không có master và slave như SPI.
* Chân Tx (truyền) của một chip kết nối trực tiếp với chân Rx (nhận) của chip kia và ngược lại. Thông thường, quá trình truyền sẽ diễn ra ở 3.3V hoặc 5V. UART là một giao thức một master, một slave, trong đó một thiết bị được thiết lập để giao tiếp với duy nhất một thiết bị khác.
* Dữ liệu truyền đến và đi từ UART song song với thiết bị điều khiển (ví dụ: CPU).
* Khi gửi trên chân Tx, UART đầu tiên sẽ dịch thông tin song song này thành nối tiếp và truyền đến thiết bị nhận.
* UART thứ hai nhận dữ liệu này trên chân Rx của nó và biến đổi nó trở lại thành song song để giao tiếp với thiết bị điều khiển của nó.
* UART truyền dữ liệu nối tiếp, theo một trong ba chế độ:

- Full duplex: Giao tiếp đồng thời đến và đi từ mỗi master và slave.

- Half duplex: Dữ liệu đi theo một hướng tại một thời điểm

- Simplex: Chỉ giao tiếp một chiều

* Dữ liệu truyền qua UART được tổ chức thành các gói. Mỗi gói chứa 1 bit bắt đầu, 5 đến 9 bit dữ liệu (tùy thuộc vào UART), một bit chẵn lẻ tùy chọn và 1 hoặc 2 bit dừng.
* **Giao tiếp UART không có chân xung nhiệp SCK cho nên sử dụng Baud rate (tốc độ baud) của 2 bên như nhau để truyền nhận. Nếu 2 bên không cùng baudrate thì sẽ gặp lỗi.**
* Frame (khung truyền): Khung truyền quy định về mỗi lần truyền bao nhiêu bit
* **Start bit: là bit đầu tiên được truyền trong 1 Frame. Báo hiệu cho thiết bị nhận có một gói dữ liệu sắp đc truyền đến. Đây là bit bắt buộc. Chân phát Tx của chủ thể 1 bình thường thì luôn ở mức cao, bit này sẽ kéo xuống mức thấp để báo hiệu cho chủ thể 2 sắp có dữ liệu được truyền qua để sẵn sàng nhận dữ liệu**
* Data: dữ liệu cần truyền. Bit có trọng số nhỏ nhất LSB được truyền trước sau đó đến bit MSB.
* **Parity bit: kiểm tra dữ liệu truyền có đúng không hay còn gọi là bit chẳn lẽ, có thể có hoặc không.**
* **Stop bit : là 1 hoặc các bit báo cho thiết brằng các bit đã được gửi xong. Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm bảo tính đúng đắn của dữ liệu. Đây là bit bắt buộc.**

**Như vậy mỗi lần truyền hoàn chỉnh thì phải có : starbit + data + (parity bit) + stop bit !**

**SPI**

* **Serial Peripheral interface: Giao tiếp ngoại vi nối tiếp**
* **Truyền dữ liệu trên cùng 1 giây, bit 1 xong đến bit 2 và cứ nối tiếp như z…**
* **Có chủ thể nằm giữa là master (thiết bị 1) và giao tiếp được với nhiều slave khác nhau…(thiết bị 2,3,4….7). thiết bị chủ toàn quyền quyết định truyền và nhận trong hệ thống này.**
* **Dữ liệu đi 2 chiều: MOSI ( chủ phát ra tớ nhận vào )**

**MISO ( tớ phát chủ nhận)**

* **Hệ thống chỉ có 1 đường MOSI và 1 đường MISO mà thôi !!!!!**

**Để trao đổi dữ liệu** thì sử dụng **đường xung nhịp nối tiếp (SCK), Master sẽ tạo ra xung nhịp trên đường SCK.**

**CPOL: Clock Polarity quyết định trạng thái lúc không hoạt động của SCK, có 2 dạng:**

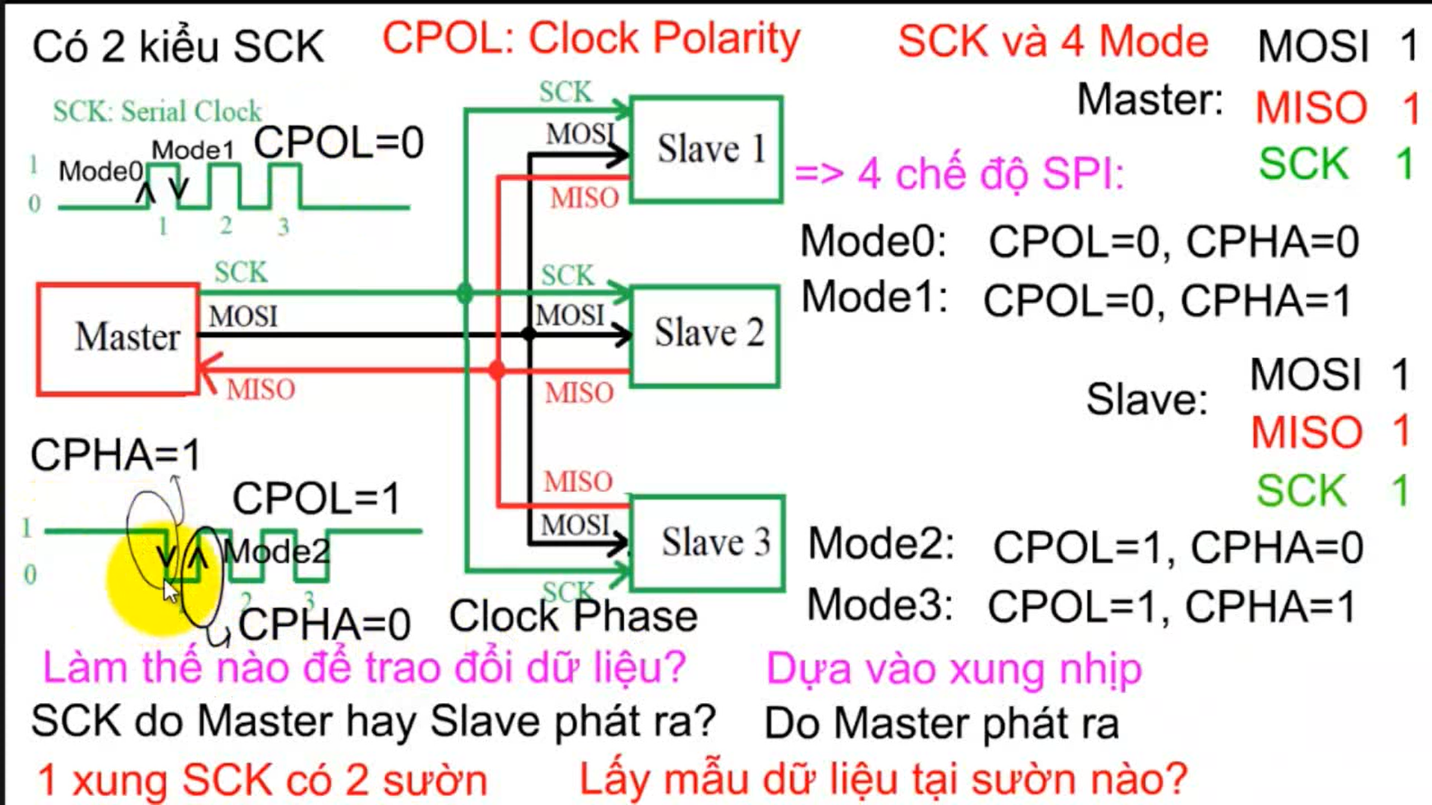
**+ dạng mức cao thì lúc tạo xung SCK sẽ tạo xung mức thấp**

**+ dạng mức thấp thì tạo xung SCK sẽ tạo xung mức cao**

**CPHA là bit quyết địhh lấy mẫu ở sườn lên hay sườn xuống của xung SCK**

**CPHA = 0, lấy mấu dữ liệu sườn lên**

**CPHA = 1, lấy mẫu dữ liệu ở sườn xuống.**

* **SPI có 4 chế độ** **(mode 0,1,2,3)**
* **Yêu cầu master phải cùng chế dộ, chế độ nào cũng được**
* **Thường thì dùng mode 0 để thực hiện giao tiếp SPI**

**Đường SS ( slave select): đường tín hiệu xác định slave nào sẽ giao tiếp với master.**

**SS thường bằng 1, đường nào được chọn thì sẽ bằng 0.**

**Tại mỗi thời điểm thì master chỉ giao tiếp với 1 slave => chỉ có 1 đường bằng 0.**

**Mỗi slave thì có 1 đường ss riêng ! master có bao nhiêu slave thì có bấy nhiêu đường ss.**