

# Lập trình C để nhúng hệ thống vi điều khiển.

*Có kinh nghiệm với lập  
trình hợp ngữ.*

VP Nelson

# Đề cương

- Tổ chức chương trình và bộ nhớ vi điều khiển
- Kiểu dữ liệu, hằng số, biến
- Địa chỉ cổng / đăng ký vi điều khiển
- Toán tử: số học, logic, shift
- Cấu trúc điều khiển: if, while, for
- Chức năng
- Các quy trình gián đoạn

# Cấu trúc chương trình C cơ bản

```
# bao gồm "STM32L1xx.h" /* Cổng I / O / tên đăng ký / địa chỉ cho vi điều khiển STM32L1xx * /
```

```
/* Biến toàn cục - tất cả các hàm đều có thể truy cập được * /
```

```
số int, bob; // biến toàn cục (tĩnh) - được đặt trong RAM
```

```
/* Định nghĩa hàm * /
```

```
int function1 (char x) // tham số x được truyền cho hàm, hàm trả về giá trị nguyên // các biến
```

```
{int i, j; // cục bộ (tự động) - được cấp phát cho ngăn xếp hoặc đăng ký
```

```
- - hướng dẫn để thực hiện chức năng
```

```
}
```

```
/* Chương trình chính */
```

```
void main (void) {
```

```
char sw1 không dấu; // biến cục bộ (tự động) (ngăn xếp hoặc đăng ký) //
```

```
int k; // biến cục bộ (tự động) (ngăn xếp hoặc đăng ký)
```

```
/* Phần khởi tạo * /
```

```
- - hướng dẫn khởi tạo biến, cổng I / O, thiết bị, thanh ghi chức năng
```

```
/* Vòng lặp vô tận */
```

```
trong khi (1) { // Cũng có thể sử dụng: for (;;) {
```

```
- - hướng dẫn được lặp lại
```

```
} /* lặp lại mãi mãi * /
```

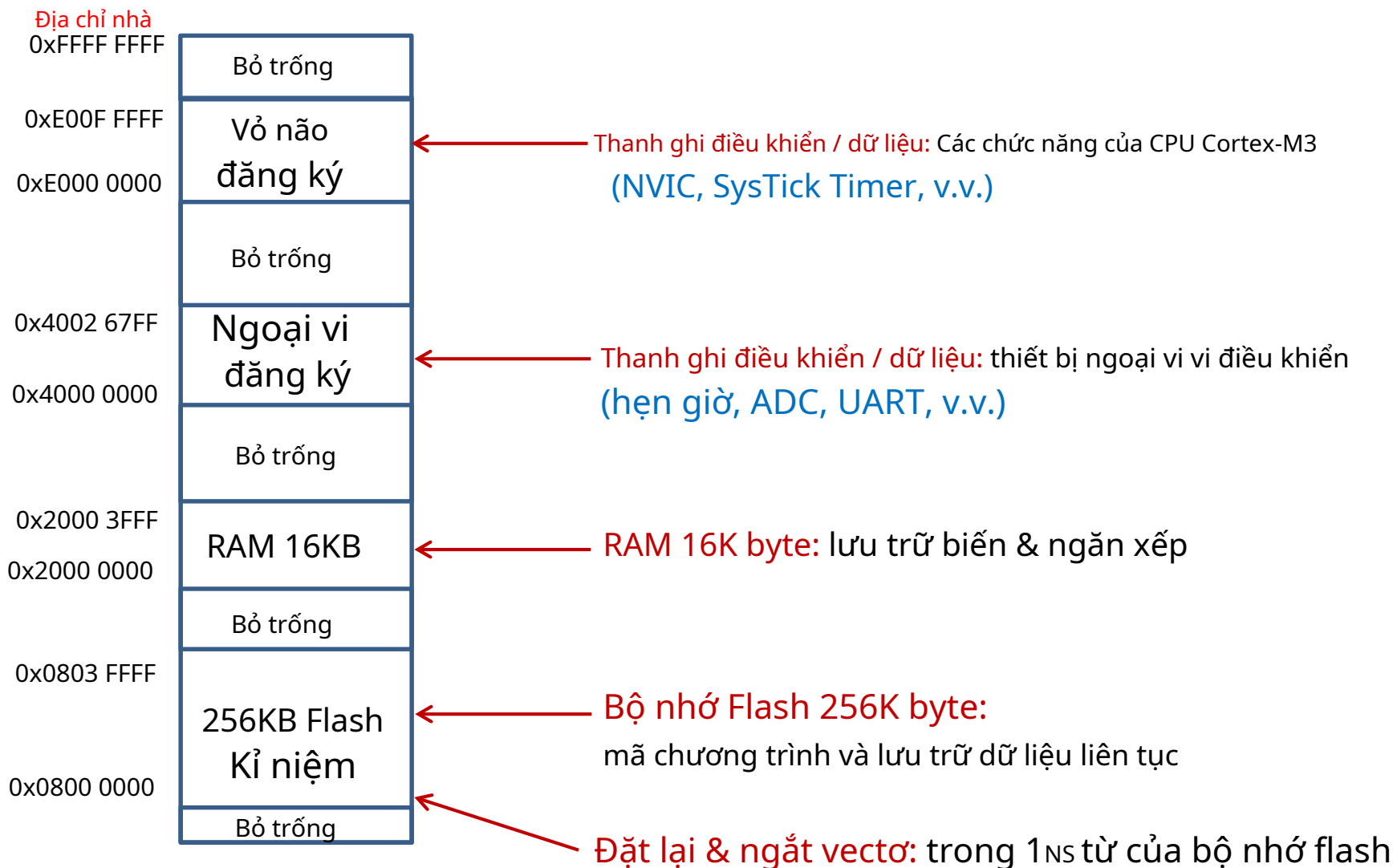
```
}
```

Khai báo các biến cục bộ

Khởi tạo các biến / thiết bị

Nội dung của chương trình

# Bản đồ bộ nhớ STM32L100RC $\mu$ C



# "Tập tiêu đề" của bộ vi điều khiển

- *Keil MDK-ARM* cung cấp một *dẫn xuất cụ thể* "tập tiêu đề" cho mỗi bộ vi điều khiển, xác định địa chỉ bộ nhớ và nhãn tượng trưng cho địa chỉ thanh ghi chức năng CPU và ngoại vi.

# bao gồm *"STM32L1xx.h"*                      */\* thông tin uC mục tiêu \*/*

*// GPIOA địa chỉ thanh ghi cấu hình / dữ liệu được xác định trong STM32L1xx.h*

*void main (void) {*

*uint16\_t PAval;*

*GPIOA-> MODER &= ~ (0x00000003);*

*PAval = GPIOA-> IDR;*

*while(1) {                      /\* thực thi mãi mãi \*/*

*}*

*// biến không dấu 16 bit*

*// Đặt chân GPIOA PA0 làm đầu vào*

*// Đặt PAval thành 16 bit từ GPIOA*

## Các kiểu dữ liệu của trình biên dịch C

- Luôn khớp kiểu dữ liệu với đặc điểm dữ liệu!
- Loại biến cho biết cách dữ liệu được biểu thị
  - # bit xác định phạm vi giá trị số
  - có dấu / không dấu xác định toán tử số học / quan hệ nào sẽ được trình biên dịch sử dụng
  - dữ liệu không phải số phải là "không dấu"
- Tập tiêu đề "stdint.h" xác định tên kiểu thay thế cho các kiểu dữ liệu C chuẩn
  - Loại bỏ sự mơ hồ liên quan đến #bits
  - Loại bỏ sự mơ hồ liên quan đến đã ký / chưa ký

(Các loại được xác định ở trang tiếp theo)

## Các kiểu dữ liệu của trình biên dịch C

| Khai báo kiểu dữ liệu *                    | Số lượng bit | Phạm vi giá trị                 |
|--|--------------|---------------------------------|
| char k;<br>char không dấu k;<br>uint8_t k; | số 8         | 0..255                          |
| ký char k;<br>int8_t k;                    | số 8         | - 128 .. + 127                  |
| k ngắn;<br>ký ngắn k;<br>int16_t k;        | 16           | - 32768 .. + 32767              |
| không dấu k ngắn;<br>uint16_t k;           | 16           | 0..65535                        |
| int k;<br>đã ký int k;<br>int32_t k;       | 32           | - 2147483648 ..<br>+ 2147483647 |
| không dấu int k;<br>uint32_t k;            | 32           | 0..4294967295                   |

\* `intx_t` và `uintx_t` được định nghĩa trong `stdint.h`

# Ví dụ về kiểu dữ liệu

- Đọc các bit từ GPIOA (16 bit, không phải số)
  - `uint16_t n; n = GPIOA->IDR;`      *// hoặc: unsigned short n;*
- Ghi giá trị tỷ lệ đặt trước TIM2 (16-bit không dấu)
  - `uint16_t t; TIM2->PSC = t;` *// hoặc: unsigned short t;*
- Đọc giá trị 32 bit từ ADC (không dấu)
  - `uint32_t a; a = ADC;`      *// hoặc: unsigned int a;*
- Phạm vi giá trị kiểm soát hệ thống [-1000... + 1000]
  - `int32_t ctrl; ctrl = (x + y) * z;` *// hoặc: int ctrl;*
- Bộ đếm vòng lặp cho 100 vòng lặp chương trình (không dấu)
  - `uint8_t cnt;`      *// hoặc: unsigned char cnt;*
  - `for (cnt = 0; cnt < 20; cnt++) {`



# Giá trị không đổi / chữ

- **Số thập phân** là định dạng số mặc định

```
int m, n;           // số có dấu 16 bit  
m = 453; n = -25;
```

- **Hệ thập lục phân**: giá trị lời nói đầu bằng 0x hoặc 0X

```
m = 0xF312; n = -0x12E4;
```

- **Hệ bát phân**: giá trị lời nói đầu bằng không (0)

```
m = 0453; n = -023;
```

Không sử dụng các số 0 ở đầu trên các giá trị "thập phân". Chúng sẽ được hiểu là bát phân.

- **Tính cách**: ký tự trong dấu ngoặc kép hoặc giá trị ASCII theo sau "dấu gạch chéo"

```
m = 'a';           // Giá trị ASCII 0x61  
n = '\ 13'; // Giá trị ASCII 13 là ký tự "return"
```

- **Dây** (mảng) các ký tự:

```
char không dấu k [7];
```

```
strcpy (m, "xin chào \ n"); // k [0] = 'h', k [1] = 'e', k [2] = 'l', k [3] = 'l', k [4] = 'o',  
                               // k [5] = 13 hoặc '\ n' (ký tự dòng mới ASCII), //  
                               k [6] = 0 hoặc '\ 0' (ký tự rỗng - cuối chuỗi)
```

# Biến chương trình

- MỘT *Biến đổi* là một vị trí lưu trữ có thể địa chỉ để thông tin được sử dụng bởi chương trình
  - Mỗi biến phải *khải báo* để chỉ ra kích thước và loại thông tin được lưu trữ, cùng với tên được sử dụng để tham chiếu thông tin

*int x, y, z; // khai báo 3 biến kiểu "int" char  
a, b; // khai báo 2 biến kiểu "char"*

- Không gian cho các biến có thể được phân bổ trong thanh ghi, RAM hoặc ROM / Flash (đối với hằng số)
- Các biến có thể là *tự động* hoặc *tĩnh*

# Mảng biến

- Một *mảng* là một tập hợp dữ liệu, được lưu trữ trong các vị trí bộ nhớ liên tiếp, bắt đầu từ một địa chỉ được đặt tên
  - Khai báo tên mảng và số phần tử dữ liệu, N
  - Các phần tử được "lập chỉ mục", với các chỉ số [0 .. N-1]

Địa chỉ nhà:

*n*

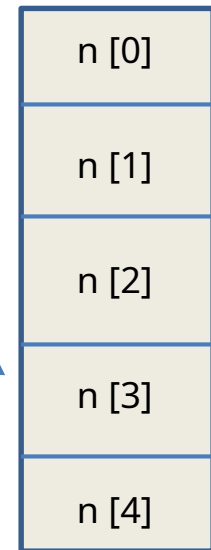
*n + 4*

*n + 8*

*n + 12*

*n + 16*

```
int n [5];           // khai báo mảng gồm 5 giá trị  
n [3] = 5;          "int" // giá trị đặt là 4ns phần tử mảng
```




Lưu ý: Chỉ số của phần tử đầu tiên luôn là 0.

# Biến tự động

- Khai báo trong một hàm / thủ tục
- Biến có thể nhìn thấy được (có *phạm vi*) chỉ trong chức năng đó
  - Không gian cho biến được phân bổ trên hệ thống cây rơm khi thủ tục được nhập
    - Đã phân bổ, được sử dụng lại, khi thủ tục được thoát
  - Nếu chỉ có 1 hoặc 2 biến, trình biên dịch có thể phân bổ chúng cho đăng ký trong thủ tục đó, thay vì cấp phát bộ nhớ.
  - Giá trị không được giữ lại giữa các lần gọi thủ tục

# Ví dụ về biến tự động

```
void delay () {  
    int i, j; // biến tự động - chỉ hiển thị trong delay ()  
    for (i = 0; i < 100; i++) { // vòng lặp ngoài  
        for (j = 0; j < 20000; j++) { // vòng  
            // không làm gì cả  
        }  
    }  
}
```



Các biến phải được khởi tạo mỗi khi thủ tục được nhập vì các giá trị không được giữ lại khi thủ tục được thoát.

*MDK-ARM (trong ví dụ của tôi): các thanh ghi được cấp phát r0, r2 cho các biến i, j*

# Biến tĩnh

- Được giữ lại để sử dụng trong suốt chương trình ở các vị trí RAM *không phân bổ lại* trong quá trình thực hiện chương trình.
- Khai báo bên trong hoặc bên ngoài một hàm
  - Nếu được khai báo bên ngoài một hàm, biến là *toàn cầu* trong phạm vi, tức là đã biết tất cả các chức năng của chương trình
    - Sử dụng khai báo "bình thường". Thí dụ: *số int;*
  - Nếu được khai báo trong một hàm, hãy chèn từ khóa *tĩnh* trước định nghĩa biến. Biến là *địa phương* trong phạm vi, tức là chỉ được biết trong chức năng này.

*char bob tĩnh không  
dấu; áp suất int tĩnh [10];*

# Ví dụ về biến tĩnh

**số lượng char không dấu;** //biến toàn cục là tĩnh - được cấp phát một vị trí RAM cố định

// count có thể được tham chiếu bởi bất kỳ hàm nào

void math\_op ()

{int i;

**int tĩnh j;**

nếu (đếm == 0)

j = 0;

i = đếm;

j = j + i;

}

// biến tự động - không gian được cấp phát trên ngăn xếp khi hàm được

nhập // biến tĩnh - được cấp phát vị trí RAM cố định để duy trì giá trị // giá trị kiểm tra của số biến toàn cục

// khởi tạo biến tĩnh j lần đầu tiên nhập vào math\_op () // khởi tạo

biến tự động i mỗi khi nhập vào math\_op () // thay đổi biến tĩnh j -

giá trị được giữ cho lần gọi hàm tiếp theo // trả về & phân bổ

không gian được sử dụng bởi biến tự động i

void main (void) {

đếm = 0;

// khởi tạo số lượng biến toàn cục

trong khi (1) {

toán\_op ();

tính ++;

// số lượng biến toàn cục tăng dần

}

# Các loại câu lệnh C

- Các phép gán biến đơn giản
  - Bao gồm truyền dữ liệu đầu vào / đầu ra
- Các phép tính toán học
- Phép toán logic / chuyển dịch
- Cấu trúc điều khiển
  - NẾU, KHI NÀO, CHO, CHỌN
- Các cuộc gọi hàm
  - Do người dùng xác định và / hoặc các hàm thư viện



# Các phép tính toán học

- Ví dụ C - với các toán tử số học tiêu chuẩn

```
int i, j, k;           // số nguyên có dấu 32 bit // số
uint8_t m, n, p;       không có dấu 8 bit
i = j + k;             // cộng các số nguyên 32 bit // trừ các
m = n - 5;             số 8 bit // nhân các số nguyên 32
j = i * k;             bit // thương của phép chia 8 bit //
m = n / p;             phần dư của phép chia 8 bit
m = n % p;
i = (j + k) * (i - 2); // biểu thức số học
```

\*, /, % có mức độ ưu tiên cao hơn +, - (mức độ ưu tiên cao hơn được áp dụng 1  
ns) Ví dụ:  $j * k + m / n = (j * k) + (m / n)$

Các định dạng dấu chấm động không được CPU Cortex-M3 hỗ trợ trực tiếp.

# Toán tử logic song song bit

Toán tử logic song song (theo chiều dọc theo bit) tạo ra kết quả n bit của phép toán logic tương ứng:

$\&$  (VÀ)       $|$  (HOẶC)       $\wedge$  (XOR)       $\sim$  (Bổ sung)

**C = A & B;  
(VÀ)**

```
MỘT 0 1 1 0 0 1 1 0
NS   1 0 1 1 0 0 1 1
-----
NS   0 0 1 0 0 0 1 0
```

**C = A | NS;  
(HOẶC)**

```
MỘT 0 1 1 0 0 1 0 0
NS   0 0 0 1 0 0 0 0
-----
NS   0 1 1 1 0 1 0 0
```

**C = A ^ B;  
(XOR)**

```
MỘT 0 1 1 0 0 1 0 0
NS   1 0 1 1 0 0 1 1
-----
NS   1 1 0 1 0 1 1 1
```

**B = ~ A;  
(BỔ SUNG)**

```
MỘT 0 1 1 0 0 1 0 0
NS   1 0 0 1 1 0 1 1
-----
```

# Đặt bit / đặt lại / bổ sung / kiểm tra

- Sử dụng “mặt nạ” để chọn (các) bit cần thay đổi

**C = A & 0xFE;**

MỘTabcdefg NS  
0xFE 1 1 1 1 1 1 1 0 Xóa bit đã chọn của A  
NS abcdefg 0

**C = A & 0x01;**

MỘTabcdefg NS  
0xFE 0 0 0 0 0 0 0 1 Xóa tất cả trừ bit đã chọn của A  
NS 0 0 0 0 0 0 0 NS

**C = A | 0x01;**

MỘTabcdefg NS  
0x01 0 0 0 0 0 0 0 1 Đặt bit đã chọn của A  
NS abcdefg 1

**C = A ^ 0x01;**

MỘTabcdefg NS  
0x01 0 0 0 0 0 0 0 1 Bổ sung bit đã chọn của A  
NS abcdefg NS'

# Ví dụ về bit cho đầu vào / đầu ra

- Tạo một "xung" trên bit 0 của PORTA (giả sử bit ban đầu là 0)

*PORTA = PORTA | 0x01; // Buộc bit 0 thành 1*

*PORTA = PORTA & 0xFE; // Buộc bit 0 thành 0*

- Ví dụ:

*if ((PORTA & 0x80) != 0) //Hoặc: ((PORTA & 0x80) == 0x80)*

*bob (); // gọi bob () nếu bit 7 của PORTA là 1*

*c = PORTB & 0x04; // che tất cả trừ bit 2 của giá trị PORTB*

*if ((PORTA & 0x01) == 0) // kiểm tra bit 0 của PORTA*

*PORTA = c | 0x01; // ghi c vào PORTA với bit 0 được đặt thành 1*

# Ví dụ về định nghĩa địa chỉ thanh ghi $\mu$ C trong *STM32Lxx.h*

*(đọc tập tiêu đề này để xem các chức năng ngoại vi khác)*

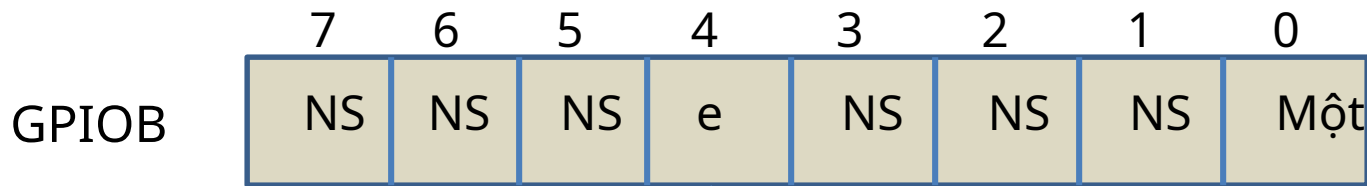
```
# xác định PERIPH_BASE      ((uint32_t) 0x40000000)      // Địa chỉ cơ sở ngoại vi trong bộ nhớ //
# xác định AHBPERIPH_BASE   (PERIPH_BASE + 0x20000)      Thiết bị ngoại vi AHB

/* Địa chỉ cơ sở của các khối thanh ghi dữ liệu / điều khiển GPIO */
# xác định GPIOA_BASE (AHBPERIPH_BASE + 0x0000) // Đăng ký GPIOA
# xác định GPIOB_BASE (AHBPERIPH_BASE + 0x0400) // Đăng ký GPIOB
# xác định GPIOA ((GPIO_TypeDef *) GPIOA_BASE) // Con trỏ tới khối thanh ghi GPIOA
# xác định GPIOB ((GPIO_TypeDef *) GPIOB_BASE) // Con trỏ tới khối thanh ghi GPIOB

/* Khoảng cách địa chỉ từ địa chỉ cơ sở GPIO - khối thanh ghi được định nghĩa là một "cấu trúc" */
typedef struct
{
    __IO uint32_t MODER;      /*! <Thanh ghi chế độ cổng GPIO,          Độ lệch địa chỉ: 0x00      */
    __IO uint16_t OTYPER;     /*! <Thanh ghi loại đầu ra cổng GPIO, / *!      Địa chỉ bù đắp: 0x04      */
    uint16_t RESERVED0;      /* <Dành riêng,                                0x06      */
    __IO uint32_t OSPEEDR; /*! <Thanh ghi tốc độ đầu ra cổng GPIO,          Địa chỉ bù đắp: 0x08      */
    __IO uint32_t PUPDR;     /*! <Thanh ghi kéo lên / kéo xuống cổng GPIO, / *!      Địa chỉ bù đắp: 0x0C      */
    __IO uint16_t IDR;        /* <Thanh ghi dữ liệu đầu vào cổng GPIO, / *! <Dành      Địa chỉ bù đắp: 0x10      */
    uint16_t ĐẶT CHỖ1;      /* riêng,                                0x12      */
    __IO uint16_t ODR;        /*! <Thanh ghi dữ liệu đầu ra cổng GPIO, /          Địa chỉ bù đắp: 0x14      */
    uint16_t RESERVED2;      /*! <Dành riêng,                                0x16      */
    __IO uint16_t BSRR;      /*! <Đặt bit cổng GPIO / đặt lại thanh ghi thấpBSRR, Độ lệch địa chỉ:      */
    __IO uint16_t BSRRH;     /* 0x18 / *! <Đặt bit cổng GPIO / đặt lại thanh ghi caoBSRR, Độ lệch địa chỉ:      */
    __IO uint32_t LCKR;      /* 0x1A / *! <Thanh ghi khóa cấu hình cổng GPIO, Độ lệch địa chỉ: 0x1C      */
    __IO uint32_t AFR [2];   /*! <Thanh ghi thấp chức năng thay thế GPIO,          Độ lệch địa chỉ: 0x20-0x24 */
} GPIO_TypeDef;
```

# Ví dụ: Các bit cổng I / O

## (sử dụng nửa dưới của GPIOB)



Chuyển kết nối với bit 4 (PB4) của GPIOB

```
uint16_t sw;
sw = GPIOB-> IDR;
sw = GPIOB-> IDR & 0x0010; // sw = 000e0000 (che tất cả trừ bit 4)

// Loại không dấu 16 bit vì GPIOB IDR và ODR = 16 bit
// sw = xxxxxxxxhgfdcb (8 bit trên từ PB15-PB8)

// Kết quả là sw = 00000000 hoặc 00010000 // KHÔNG BAO GIỜ ĐÚNG cho sw trên, là 000e0000 // TRUE nếu e = 1 (bit 4 trong kết quả của PORTB & 0x10) // TRUE nếu e = 0 trong PORTB & 0x10 (sw = 00000000) // TRUE nếu e = 1 trong PORTB & 0x10 (sw = 00010000) // Ghi vào 16 bit của GPIOB; kết quả là 01011010 // Chỉ đặt bit e thành 1 trong GPIOB (GPIOB bây giờ là hg1f1dcba) // Chỉ đặt lại bit e thành 0 trong GPIOB (GPIOB bây giờ là hg0f0dcba)

nếu (sw == 0x01)
if (sw == 0x10)
nếu (sw == 0)
nếu (sw != 0)
GPIOB-> ODR = 0x005a;
GPIOB-> ODR |= 0x10;
GPIOB-> ODR &= ~0x10;
if ((GPIOB-> IDR & 0x10) == 1) // TRUE nếu e = 1 (bit 4 của GPIOB)
```

# Các toán tử thay đổi

Toán tử Shift:

$x \gg y$  (dịch phải toán hạng  $x$  theo vị trí bit  $y$ )

$x \ll y$  (dịch trái toán hạng  $x$  theo vị trí bit  $y$ ) Các bit trống được điền bằng 0.

Chuyển nhanh sang phải / trái để nhân / chia theo lũy thừa của 2

**B = A << 3;**  
(Dịch trái 3 bit)

MỘT 1 0 1 0 1 1 0 1  
NS 0 1 1 0 1 0 0 0

**B = A >> 2;**  
(Dịch sang phải 2 bit)

MỘT 1 0 1 1 0 1 0 1  
NS 0 0 1 0 1 1 0 1

**B = '1';**  
**C = '5';**  
**D = (B << 4) | (C & 0x0F);**  
(B << 4)  
(C & 0x0F)

**B = 0 0 1 1 0 0 0 1 (ASCII 0x31)** **C = 0 0 1 1 0 1 0 1 (ASCII 0x35)**

NS  
= 0 0 0 1 0 0 0 0  
= 0 0 0 0 0 1 0 1  
= 0 0 0 1 0 1 0 1 (Đóng gói BCD 0x15)

# C cấu trúc điều khiển

- Kiểm soát thứ tự thực hiện các lệnh (luồng chương trình)
- Thực hiện có điều kiện
  - Thực thi một tập hợp các câu lệnh nếu một số điều kiện được đáp ứng
  - Chọn một tập hợp các câu lệnh sẽ được thực thi từ một số tùy chọn, tùy thuộc vào một hoặc nhiều điều kiện
- Thực hiện lặp đi lặp lại
  - Thực hiện lặp đi lặp lại một tập hợp các câu lệnh
    - Một số lần được chỉ định, hoặc
    - Cho đến khi một số điều kiện được đáp ứng, hoặc
    - Trong khi một số điều kiện là đúng

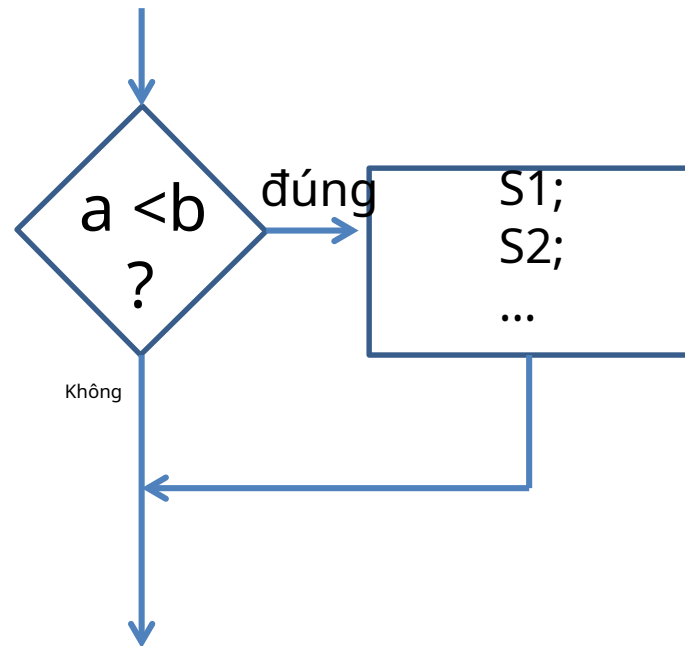


# Cấu trúc IF-THEN

- Thực thi một tập hợp các câu lệnh nếu và chỉ khi một số điều kiện được đáp ứng

Điều kiện TRUE / FALSE

```
if (a < b)
{
    câu lệnh s1;
    câu lệnh s2;
    ....
}
```



# Toán tử quan hệ

- Kiểm tra mối quan hệ giữa hai biến / biểu thức

| Thử nghiệm | Điều kiện ĐÚNG            | Ghi chú |
|------------|---------------------------|---------|
| (m == b)   | m bằng b                  | Đôi =   |
| (m != b)   | m không bằng              |         |
| (m < b)    | bm nhỏ hơn b              | 1       |
| (m <= b)   | m nhỏ hơn hoặc bằng       | 1       |
| (m > b)    | bm lớn hơn b              | 1       |
| (m >= b)   | m lớn hơn hoặc bằng b 1 m |         |
| (NS)       | khác không                |         |
| (1)        | luôn luôn đúng            |         |
| (0)        | luôn SAI                  |         |

1. Sử dụng trình biên dịch  
đã ký hoặc chưa ký  
so sánh, trong  
phù hợp với  
Loại dữ liệu

## Thí dụ:

*char không dấu a, b;  
int j, k;  
if (a < b) - unsigned  
if (j > k) - sign*

# Toán tử boolean

- Toán tử boolean **&&** (Và và **||** (HOẶC) tạo ra kết quả TRUE / FALSE khi kiểm tra nhiều điều kiện TRUE / FALSE

*if ((n > 1) && (n < 5)) // kiểm tra n từ 1 đến 5*

*if ((c == 'q') || (c == 'Q')) // kiểm tra c = chữ thường hoặc chữ hoa Q*

- Lưu ý sự khác biệt giữa **Boolean** toán tử **&&**, **||** và **logic bitwise** toán tử **&**, **|**

*nếu (k && m) // kiểm tra nếu k và m đều TRUE (các giá trị khác 0)*

*nếu (k & m) // tính theo bitwise AND giữa m và n, //sau đó kiểm tra xem kết quả có khác 0 hay không (TRUE)*

# Lỗi chung

- Lưu ý rằng `==` là một toán tử quan hệ, trong khi `=` là một toán tử gán.

*nếu ( $m == n$ ) // kiểm tra sự bình đẳng của các giá trị của biến  $m$  và  $n$*

*nếu ( $m = n$ ) // chỉ định giá trị của  $n$  thành biến  $m$ , và sau đó*

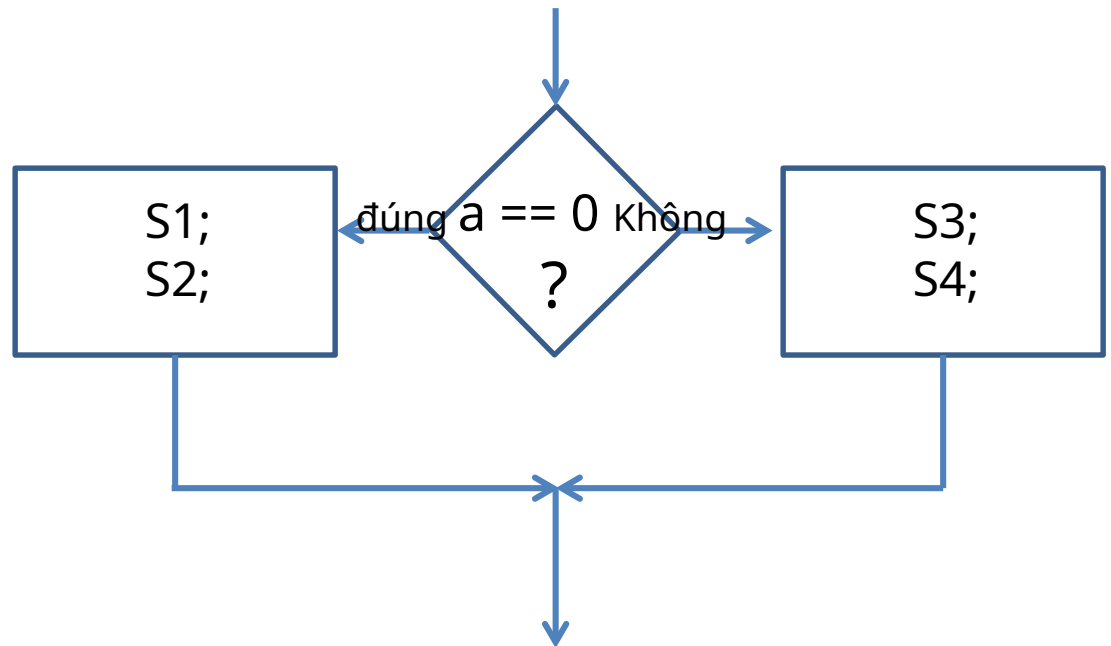
*// kiểm tra xem giá trị đó có phải là TRUE (khác không)*

Dạng thứ hai là một lỗi phổ biến (bỏ qua dấu bằng thứ hai) và thường tạo ra kết quả không mong muốn, cụ thể là điều kiện TRUE nếu  $n$  là 0 và FALSE nếu  $n$  khác 0.

# Cấu trúc IF-THEN-ELSE

- Thực thi một tập hợp các câu lệnh nếu một điều kiện được đáp ứng và một tập hợp thay thế nếu điều kiện không được đáp ứng.

```
if (a == 0)
{
    câu lệnh s1;
    câu lệnh s2;
}
khác
{
    câu lệnh s3;
    câu lệnh s4;
}
```



# Ví dụ về hợp ngữ IF-THEN-ELSE HCS12 so với C

AD\_PORT: EQU \$ 91; Cổng dữ liệu A / D  
MAX\_TEMP: EQU 128; Nhiệt độ tối đa EQU 0;  
VALVE\_OFF: Bits cho van tắt  
VALVE\_ON: EQU 1; Bits cho van trên EQU  
VALVE\_PORT: \$ 258; Cổng P cho van

...

**; Nhận nhiệt độ**

ldaa AD\_PORT

**; NẾU Nhiệt độ > Tối đa Cho phép**

cmpa #MAX\_TEMP

bis ELSE\_PART

**; SAU ĐÓ, tắt van nước**

ldaa VALVE\_OFF

staa VALVE\_PORT

áo ngược END\_IF

**; ELSE Bật van nước**

ELSE\_PART:

ldaa VALVE\_ON

staa VALVE\_PORT

END\_IF:

**; KẾT THÚC NẾU nhiệt độ > Tối đa cho phép**

Phiên bản C:

# xác định MAX\_TEMP 128

# xác định VALVE\_OFF 0

# xác định VALVE\_ON 1

nếu (AD\_PORT <= MAX\_TEMP)

VALVE\_PORT = VALVE\_OFF;

khác

VALVE\_PORT = VALVE\_ON;

# Liên kết ELSE không rõ ràng

*nếu ( $n > 0$ )*

*nếu ( $a > b$ )*

*$z = a;$*

*khác*

*// else đi với gần nhất trước “if” ( $a > b$ )*

*$z = b;$*

*nếu ( $n > 0$ ) {*

*nếu ( $a > b$ )*

*$z = a;$*

*} khác {*

*// else đi với “if” đầu tiên ( $n > 0$ )*

*$z = b;$*

*}*

Dấu ngoặc nhọn buộc liên kết thích hợp

# Nhiều cấu trúc ELSE-IF

- Quyết định đa chiều, với các biểu thức được đánh giá theo thứ tự cụ thể

*nếu ( $n == 1$ )*

*câu lệnh1; // làm nếu  $n == 1$*

*khác nếu ( $n == 2$ )*

*câu lệnh2; // làm nếu  $n == 2$*

*khác nếu ( $n == 3$ )*

*câu lệnh3; // làm nếu  $n == 3$*

*khác*

*câu lệnh4; // thực hiện nếu có bất kỳ giá trị nào khác của  $n$  (không có giá trị nào ở trên)*

Bất kỳ “câu lệnh” nào ở trên đều có thể được thay thế bằng một tập hợp các câu lệnh: {s1; s2; s3; ...}



# Tuyên bố SWITCH

- Thay thế nhỏ gọn cho cấu trúc ELSE-IF, cho quyết định đa đường kiểm tra một biến hoặc biểu thức cho một số giá trị không đổi

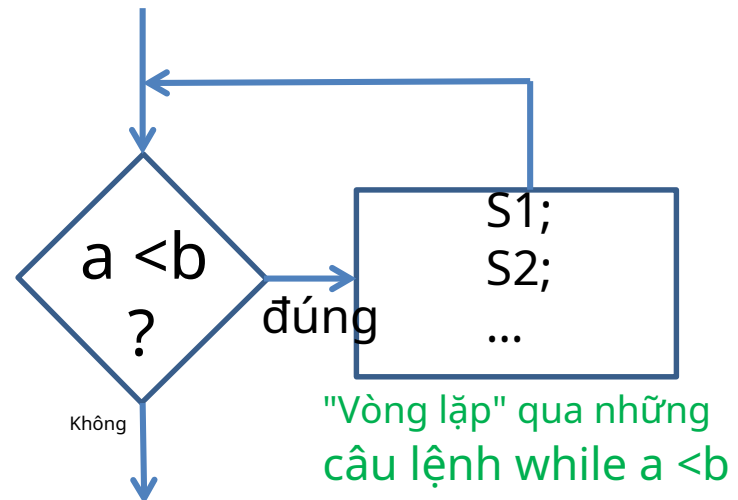
*/ \* ví dụ tương đương với ví dụ trên trang trình bày trước đó \* / switch (n) { // n là biến được kiểm tra  
    case 0: statement1; // do if n == 0  
    case 1: statement2; // do if n == 1  
    case 2: statement3; // làm nếu n == 2  
    default: statement4; // nếu cho bất kỳ giá trị n nào khác  
}*

Bất kỳ “câu lệnh” nào ở trên đều có thể được thay thế bằng một tập hợp các câu lệnh: {s1; s2; s3; ...}

# Cấu trúc vòng lặp WHILE

- Lặp lại một tập hợp các câu lệnh (một “vòng lặp”) miễn là như một số điều kiện được đáp ứng

```
while (a < b)  
{  
    câu lệnh s1;  
    câu lệnh s2;  
    ....  
}
```



Một cái gì đó cuối cùng phải gây ra  $a \geq b$ , để thoát khỏi vòng lặp

# Ví dụ về vòng lặp WHILE:

## Ngôn ngữ hợp ngữ C so với HCS12

### Phiên bản C:

```
# xác định MAX_ALLOWED 128
# xác định LIGHT_ON 1
# xác định LIGHT_OFF 0
```

```
trong khi (AD_PORT <= MAX_ALLOWED) {
```

```
    LIGHT_PORT = LIGHT_ON;
```

```
    trì hoãn();
```

```
    LIGHT_PORT = LIGHT_OFF;
```

```
    trì hoãn();
```

```
}
```

```
QUẢ_NG CÁO: EQU $ 91; A / D Cổng dữ liệu
MAX_ALLOWED: EQU 128; Nhiệt độ tối đa
LIGHT_ON: EQU 1
LIGHT_OFF: EQU 0
LIGHT_PORT: EQU $ 258; Cổng
P; - - -
```

```
; Nhận nhiệt độ từ A / D
```

```
    ldaa AD_PORT
```

```
; KHI nhiệt độ > tối đa cho phép
```

```
WHILE_START:
```

```
    cmpa MAX_ALLOWED
```

```
    bls END_WHILE
```

```
; NÊN - Đèn flash bật 0,5 giây, tắt 0,5 giây
```

```
    ldaa LIGHT_ON
```

```
    staa LIGHT_PORT; Biến độ trễ jsr
```

```
    ánh sáng; Độ trễ 0,5 giây
```

```
    ldaa LIGHT_OFF
```

```
    staa LIGHT_PORT; Tắt đèn trễ jsr
```

```
; Kết thúc nhấp nháy đèn, Nhận nhiệt độ từ A / D
```

```
    ldaa AD_PORT
```

```
; END_DO
```

```
áo lót    WHILE_START
```

```
END_WHILE:
```

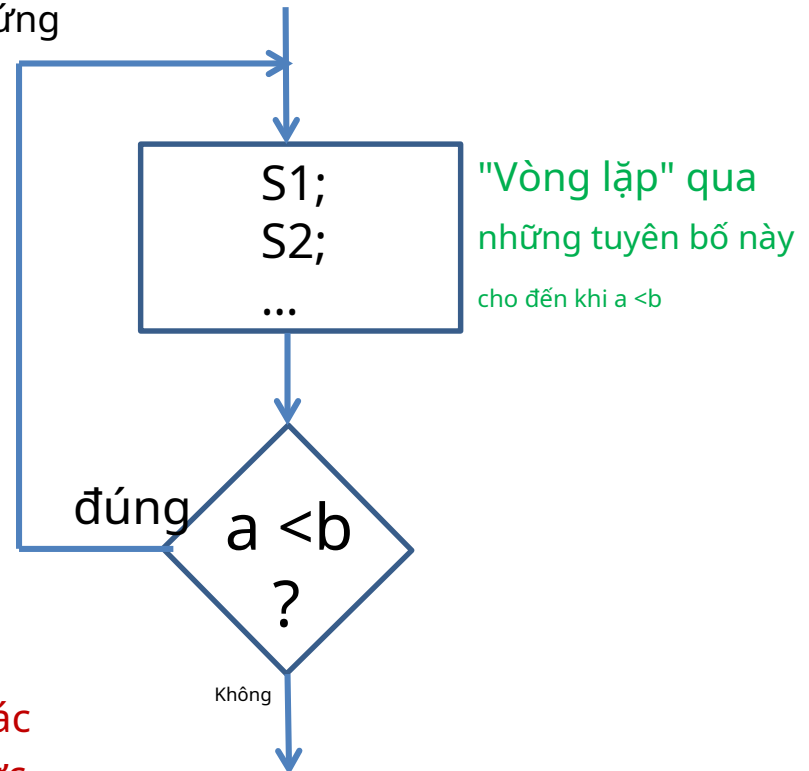
# Cấu trúc vòng lặp DO-WHILE

- Lặp lại một tập hợp các câu lệnh (một “vòng lặp”)

**cho đến khi** một số điều kiện được đáp ứng

```
làm  
{  
    câu lệnh s1;  
    câu lệnh s2;  
    ....  
}  
while (a < b);
```

Điều kiện được kiểm tra sau thực thi tập hợp các câu lệnh, vì vậy các câu lệnh được đảm bảo thực thi ít nhất một lần.



# Ví dụ DO-WHILE

## Phiên bản C:

# xác định MAX\_ALLOWED 128

# xác định LIGHT\_ON 1

# xác định LIGHT\_OFF 0

làm {

    LIGHT\_PORT = LIGHT\_ON;

    trì hoãn();

    LIGHT\_PORT = LIGHT\_OFF; trì

    hoãn();

} while (AD\_PORT <= MAX\_ALLOWED);

## **; Phiên bản hợp ngữ HCS12; LÀM**

; Đèn flash bật 0,5 giây, tắt 0,5 giây

ldaa        BẬT ĐÈN LÊN

staa        LIGHT\_PORT; Bật đèn trễ; Độ trễ

jsr        0,5 giây

ldaa        TẮT ĐÈN

staa        LIGHT\_PORT; Tắt đèn trễ

jsr

; Kết thúc nhấp nháy đèn

; Nhận nhiệt độ từ A / D

ldaa AD\_PORT

**; END\_DO**

áo lót

**WHILE\_START**

**; END\_WHILE:**

; Nhiệt độ END\_WHILE > tối đa cho phép

; Chương trình con giả

trì hoãn: rts

# Ví dụ về WHILE

*/\* Thêm hai mảng 200 phần tử. \*/*

int M [200], N [200], P [200];

int k;

*/\* Phương pháp 1 - sử dụng DO-WHILE \*/*

k = 0; *// khởi tạo bộ đếm / chỉ mục*

làm {

    M [k] = N [k] + P [k]; *// thêm phần tử mảng thứ*

    k = k + 1; *k // bộ đếm tăng / chỉ số //*

} while (k < 200); *lặp lại nếu k nhỏ hơn 200*

*/\* Phương pháp 2 - sử dụng vòng lặp WHILE*

k = 0; *// khởi tạo bộ đếm / chỉ số // thực hiện*

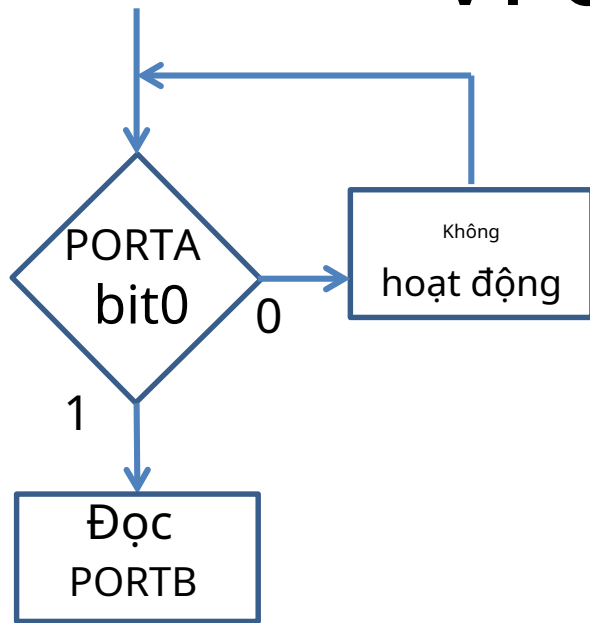
while (k < 200) { M [k] *vòng lặp nếu k nhỏ hơn 200 // thêm thứ*

    = N [k] + P [k]; k = *k phần tử mảng*

    k + 1; *// bộ đếm / chỉ số tăng*

}

# Ví dụ WHILE



Chờ 1 được áp dụng cho  
bit 0 của GPIOA  
và sau đó đọc GPIOB

*trong khi ((GPIOA-> IDR & 0x0001) == 0) // kiểm tra bit 0 của GPIOA*

*{*

*c = GPIOB-> IDR;*

*// không làm gì cả & lặp lại nếu bit là 0*

*// đọc GPIOB sau bit trên = 1*

# Cấu trúc vòng lặp FOR

- Lặp lại một tập hợp các câu lệnh (một “vòng lặp”) trong khi một số điều kiện được đáp ứng
  - thường là một số lần lặp lại nhất định

The diagram illustrates the components of a do-while loop. At the top, three labels with arrows point to parts of the loop syntax: 'Khởi tạo' (Initialization) points to the variable declaration, 'Điều kiện cho' (Condition) points to the condition, and '(Các) hoạt động khi kết thúc của mỗi vòng lặp' ((The) actions when ending each loop) points to the body of the loop. The loop syntax is shown as: `while (m = 0; m < 200; m++) {`. Below this, the body of the loop is shown as:  `câu lệnh s1;  
 câu lệnh s2;  
}`.

Khởi tạo

Điều kiện cho

(Các) hoạt động khi kết thúc của mỗi vòng lặp

`while (m = 0; m < 200; m++) {`

`câu lệnh s1;`

`câu lệnh s2;`

`}`



# Cấu trúc vòng lặp FOR

- Vòng lặp FOR là một dạng nhỏ gọn hơn của cấu trúc vòng lặp WHILE

*/\* thực hiện vòng lặp 200 lần \*/ // \* vòng lặp WHILE tương đương \* /*

*vì (m = 0; m < 200; m++) {*

*câu lệnh s1;  
    câu lệnh s2;  
}*

*m = 0; // (các) hành động ban đầu*

*while (m < 200) // kiểm tra điều kiện*

*{  
    câu lệnh s1;  
    câu lệnh s2;*

*m = m + 1; // kết thúc hành động của vòng lặp*

*}*

# Ví dụ về cấu trúc FOR

```
/* Đọc 100 giá trị 16 bit từ GPIOB vào mảng C */  
/* Bit 0 của GPIOA (PA0) là 1 nếu dữ liệu đã sẵn sàng và 0 nếu không */  
uint16_t c [100];  
uint16_t k;  
  
for (k = 0; k < 200; k++) {  
    while ((GPIOA-> IDR & 0x01) == 0) // lặp lại cho đến khi PA0 = 1  
        {}                               // không làm gì nếu PA0 = 0 //  
    c[k] = GPIOB-> IDR;                  // đọc dữ liệu từ PB [15: 0]  
}
```

# Ví dụ về cấu trúc FOR

*/ \* Vòng lặp FOR lồng nhau để tạo độ trễ thời gian \* /*

```
for (i = 0; i < 100; i++) {           // thực hiện vòng lặp bên ngoài 100 lần  
  for (j = 0; j < 1000; j++) { // thực hiện vòng lặp bên trong 1000 lần  
    }                                // không làm gì trong vòng lặp bên trong  
  }
```

# Hàm C

- Chức năng phân vùng các chương trình lớn thành một tập hợp các tác vụ nhỏ hơn
  - Giúp quản lý độ phức tạp của chương trình
  - Các tác vụ nhỏ hơn dễ thiết kế và gỡ lỗi hơn
  - Các chức năng thường có thể được sử dụng lại thay vì bắt đầu lại
  - Có thể sử dụng “thư viện” các hàm do 3 phát triển<sub>rd</sub> các bữa tiệc, thay vì thiết kế của riêng bạn


# Hàm C

- Một chức năng được "gọi" bởi một chương trình khác để thực hiện một tác vụ
  - Các *hàm số có thể* trả về một kết quả cho người gọi
  - Một hoặc nhiều đối số có thể được chuyển cho hàm / thủ tục

# Định nghĩa hàm

Loại giá trị được trả lại cho người gọi \*

Các thông số đã vượt qua bởi người gọi



```
int math_func (int k; int n)  
{  
    int j;                // biến cục bộ  
    j = n + k - 5;        // nội dung hàm  
    return (j);           // trả về kết quả  
}
```

\* Nếu không có giá trị trả về, hãy chỉ định "void"


# Đổi số hàm

- Chương trình gọi có thể chuyển thông tin đến một hàm theo hai cách
  - Qua **giá trị**: truyền một hằng số hoặc một giá trị biến
    - hàm có thể sử dụng, nhưng không thể sửa đổi giá trị
  - Qua **thăm quyền giải quyết**: truyền địa chỉ của biến
    - hàm có thể vừa đọc vừa cập nhật biến
  - Giá trị / địa chỉ thường được chuyển đến hàm bằng cách đẩy chúng vào hệ thống **cây rơm**
    - Hàm lấy thông tin từ ngăn xếp

# Ví dụ - chuyển theo giá trị

*/\* Hàm tính  $x^2$  \*/*

```
int square ( int x ) { // giá trị được truyền là kiểu int, trả về giá trị int  
    int y; // biến cục bộ - phạm vi giới hạn ở hình vuông //  
    y = x * x; sử dụng giá trị đã truyền  
    return (x); // trả về kết quả  
}
```



```
void main {  
    int k, n; // biến cục bộ - phạm vi giới hạn cho main  
    n = 5;  
    k = hình vuông (n); // truyền giá trị của n, gán n bình phương cho k  
    n = hình vuông (5); // truyền giá trị 5, gán 5 bình phương cho n  
}
```



# Ví dụ - chuyển qua tham chiếu

*/\* Hàm tính x<sub>2</sub> \*/*

```
hình vuông rỗng ( int x, int * y ) { // giá trị của x, địa chỉ của y
    * y = x * x; // ghi kết quả vào vị trí có địa chỉ là y
}
```

```
void main {
    int k, n; // biến cục bộ - phạm vi giới hạn cho main
    n = 5;
    Quảng trường(n, & k); // tính toán bình phương n và đặt kết quả là k
    Quảng trường(5 & n); // tính toán 5 bình phương và đặt kết quả là n
}
```

Ở trên, *chủ chốt* kể *Quảng trường* vị trí của biến cục bộ của nó, để *Quảng trường* có thể ghi kết quả vào biến đó.

# Ví dụ - nhận các byte dữ liệu nối tiếp

*/\* Đặt chuỗi các byte SCI đã nhận vào một mảng \*/*

```
Int rcv_data [10];           // mảng biến toàn cục cho dữ liệu đã  
Int rcv_count;               nhận // biến toàn cục cho # byte nhận được
```

```
void SCI_receive () {  
    trong khi ((SCISR1 & 0x20) == 0) {} // đợi dữ liệu mới (RDRF = 1)  
    rcv_data [rcv_count] = SCIDRL; // byte sang mảng từ SCI dữ liệu reg.  
    rcv_count ++;                  // cập nhật chỉ mục cho byte tiếp theo  
}
```

Các hàm khác có thể truy cập dữ liệu nhận được từ mảng biến toàn cục rcv\_data [].

## Một số hướng dẫn trực tuyến về C

- <http://www.cprogramming.com/tutorial/ctutorial.html>
- [http://www.physics.drexel.edu/courses/Comp\\_Phys / General / C\\_basics /](http://www.physics.drexel.edu/courses/Comp_Phys / General / C_basics /)
- <http://www.iu.hio.no/~mark/CTutorial/CTutorial.html>
- <http://www2.its.strath.ac.uk/courses/c/>

# Hướng dẫn sẽ được tiếp tục... ..