**INTERVIEW– GOOD LUCK**

# 1 Introduce yourself (confident and prepare detail)

My name is Quang, I’m graduate at Hanoi University of Industry with major is Electronic and telecommunication. Today I apply for embedded (engineer )software fresher in Bosch.

Summary about my experience:

I have 9 months as fresher position in HCL. After join HCL I have some training such as English training, technical training include Embedded C, Microcontroller stm32f4, Automotive domain, Misra rule, testing basic and software develop life cycle with trainer in Indian.

Reason I apply (FPT) Bosch company because I want to find a new environment to improve my knowledge related to embedded field especially Automotive field and follow to my career -path is will come to expert in embedded field. 2th reason I want to find a fine job because when I work in HCL company I done training on 30/April/2022 but still now I don’t have any actual Project. In that time I try to self-learn new knowledge and practice the old knowledge to avoid forget it.

I know FPT company have office on 26 countries in the world. Have 3 main business sector are Techonlogy, telecommunication, and education. Technology is an important strategic.as i know Fpt also work with top the Big automotive company such as Ford..

I know Bosch is Global company have office on the 60 countries in worldwide.

Head office at Germany, Bosch is one of the top Company technology in the world especially is embedded field, I’m sure Bosch is perfect environment to me develops career-path.

I hope if I have an opportunity I could contribute a little bit for (FPT)Bosch company

And try hard to compete all assigned task.

2 technical questions:

# **C PROGRAMING**

1. What is function pointer?(Con trỏ hàm)

* Function can point to any functions.
* Function pointer point to code not data, it store the start of excecute code.
* A function pointer can be passed as an agument or also can be return from a fuction.

1. In case use function or macro?

* Function used to hight complex problem
* Macro use to simple problem
* Function are complied
* Macro is preprocessor
* Macro no type checking so it difficult to debug
* Macro increases the size code
* Fucntion keep the code length unaffected
* Macro speed exercution is faster
* Function speed exercution is lower
* Macro Before compilation, macro name replace by macro value

## how to calculate memory assumption(giả thiết) for Struct ?

* size of struct=sum of all data members.

## Static function?

* Scope: only visible in file where they are declared
* Use it when you want to restrict access to functions.

## Should declare multiple functions in the program?

* No, becauce will lead speed to execute program slower,

## What happens when we call a function?

Understand:

**Program stack**: hold all the function calls,

**Stack frame**: a buffer memory that is an element of program stack, has data of the called func: return address, input paramenter, local variable, register savings.

**Stack pointer**: pointer point the top of program stack

**Series of operations when we call a function:**

+Stack frame is pushed in to stack.

+Sub-routine instructions are executed.

+Stack frame is poped from the stack.

+Program counter is holding the return address**.**

Link: [What happens when we call a Function - GeeksforGeeks](https://www.geeksforgeeks.org/what-happens-when-we-call-a-function/)

## Purpose keyword inline in C function?

* Inline functions are those function whose definitions are small and be replaced at the place where its function call is happened.
* Reduce time to execute program if function is small. If function is complex and big it will lead increases size code.
* Some case inline keyword is dismiss (EX: inline function include loops, static variable, recursive, goto command is switch, have type different void but don’t have return command)

## Mission of file.h? (nhiem vu)

* Contain function prototype
* Data type definitions
* Macros

2 types header file:

Pre-existing header file (stdio.h, stdint.h)

User-defined header file.

Static variables in function?

* Local static variable remains in memory while the program is running. Normal or auto variables will destroyed when function call end.
* Static variables are allocated memory in data segment, not stack segment.
* Auto initialized 0 if not initialized explicitly.
* Static variable should not be declared inside struct.

What is conditional directive?

* It is if else have hashtag (#) on the front.(#ifndef, #elif, #ifdef, #endif)

Help to compilation portion of the program or let us skip compilations of some specific part of the program

#ifndef directive: to avoid include a header file multiple times into a file.

Leads to compiler error.

Extern keyword?

* a global variable declare extern keyword common use in 2 file another.

## Difference between declaration vs definition(difference with function)?

* **Declaration** is for informing the compiler of the name, return type and initial value (memory will not be allocated during declaration)
* **Definition** says where the variable gets stored. (memory will be allocated)

## Pointer operator P=P+1?

* Point to next memory cell,

## Volatile keyword in C?

* Declare a variable with volatile key word to inform the compiler this variable can’t optimine. Because this variable can be change by interrupt, hardware… compiler can’t see change in the code.

## Why not use recursion in embedded program?

* Because when use recursion, the previous function does not release(giải phóng) so wait until the last function done to release it in turn.

## Compiler process?

* Pre-processor(.i file) -> compiler(.s file) -> assembler(.o file) -> Linking(.out )executable file
* Pre-processor: + remove the comments from source code

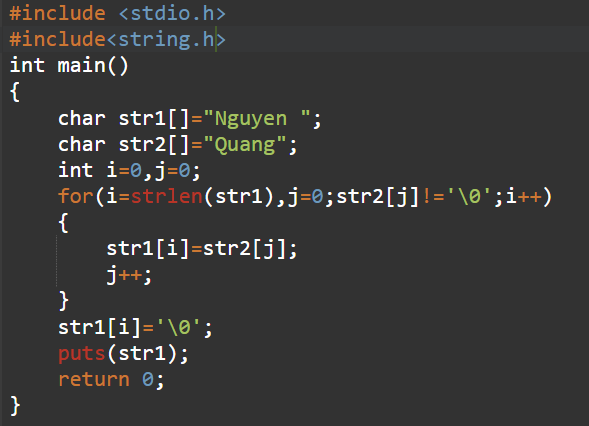
+ perform the Macro expansion if any Macro is used

+ perform the expansion of the include header files.

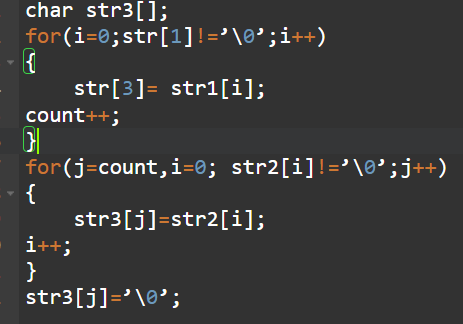
* The compiler step will take .i file and generate to assembly code (.s file )
* The assembly step will take .s file and generate to object code (.o file)
* The Linking : in the project with several(một số) modules, we will have several object file after step 3. All of these files have to be rearranged and all the missing instruction (if you using libraries) must be linked together

## How to concatenate 2 strings?

* Code: option1



* Code option2:



Padding, alignment?

* The processor does not read 1 byte at a time. It reads 1 word bytes at the time.
* 1 word byte mean: if 32 bit processor 1 Word=4 bytes

If 64-bit processor 1 word=8 bytes

* Padding : insert empty bit to the memory, so the variable can be accessed in a single CPU cycle
* EX:

**struct** student

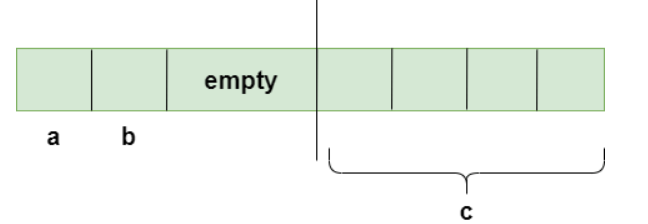
{

**char** a; // 1 byte

**char** b; // 1 byte

**int** c; // 4 bytes

}  // size of struct is 8 bytes have 2 bit padding

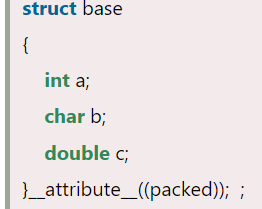


1 CPU cycle is 4 byte (32-bits processor)

## How to reduce bit padding in structure?

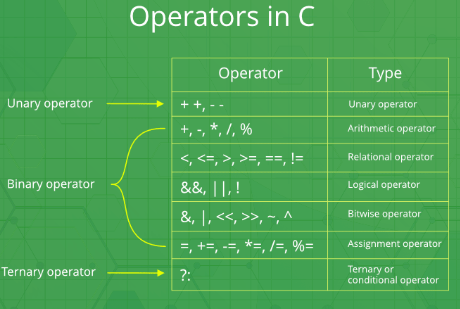
* Three way are: + using #pragma pack(1) directive

+ using attribute

(13 bytes)

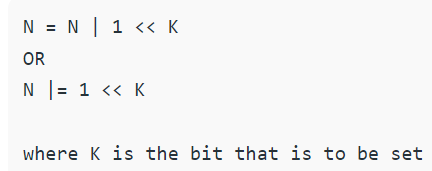
+ should be arrange suitable members in structure  increasing/decreasing order of size.

## Bitwise operator?

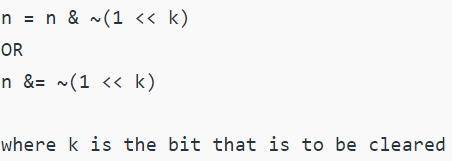


## Set, clear and toggle a given bit of a number in C

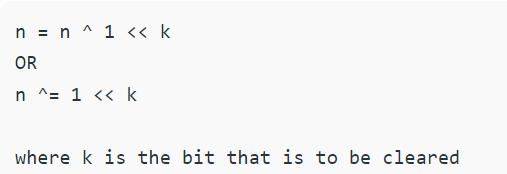
* Set bit: OR operator



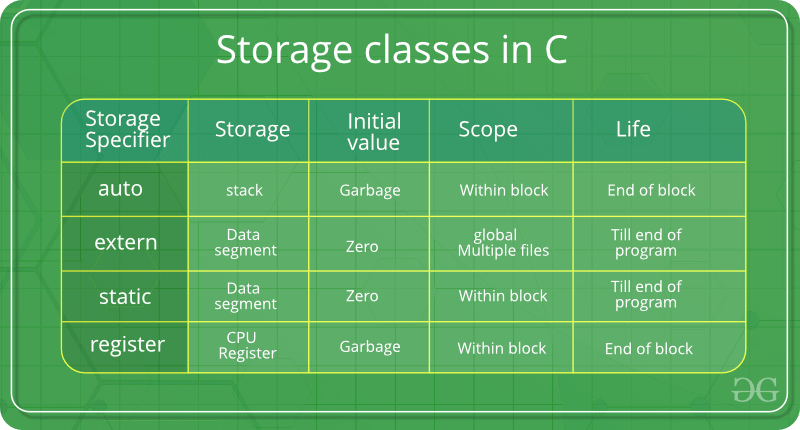
* Clear bit: AND operator



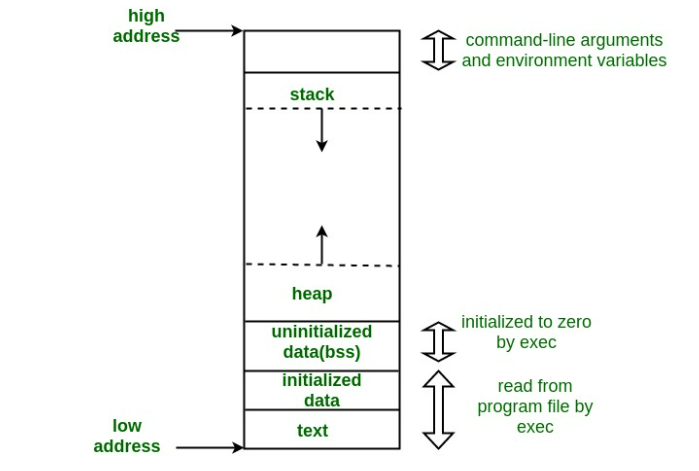
* Toggle bit: XOR operator



Storage class in C?



Memory layout in C?



* **Text segment**: storage code or simply as text. Text segment is often read only, to prevent a program from accidentally modifying its instructions.
* **Initialized data segment:** contain global or static variable that are initialized by programmer.
* **Uninitialized data segment (BSS):** contain all global variables and static variables that are initialized to 0 or not have explicit initialization.
* **Stack:** where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack. (Vùng nhớ tự động cấp phát, sau khi kết thúc chương trình vùng nhớ sẽ tự động giải phóng )
* **Heap:** Heap is the segment where dynamic memory allocation usually takes place. The Heap area is managed by malloc, realloc, and free(vùng nhớ do lập trình viên yêu cầu cấp phát và tồn tại cho đến khi coder free(), nếu k free sẽ dẫn đến hiện tượng memory leak)

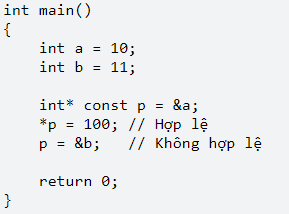
Size of pointer?

* Each operating system size of pointer is difference. 4 bytes with OS 32-bit, 8 bytes with OS 64-bits

Constant pointer? Hằng con trỏ

* The pointer to fixed memory location, the value at the location can be changed, but pointer will always point to the same location

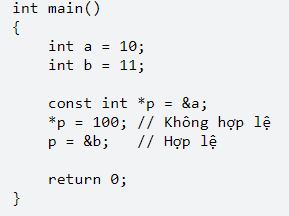
***<data type> \* const <name>***



## Pointer to constant? Con trỏ hằng

* The data pointed by the pointer is constant and can not be change. But the pointer itself can change and point somewhere

***Const <data type> \* <name>***



## Void pointer?

* Pointer not point any data type, a void pointer can hold address of any type and can be typecasted to any type.
* Void pointer Unspecific data type so can’t be access value of pointer via dereference. Need to typecast data type if you want to access value pointer

## Difference between Typedef and Define in C?

* **Typedef** is a keyword provide some meaningful names easy to remember to the already existing data type in C programing.
* **Define** used to redefinitionfor the value or constant value
* **Typedef** perform by compiler
* **Define** perform by pre-processor

## Call by value vs call by reference ?

|  |  |
| --- | --- |
| **Call by Value** | **Call by Reference** |
| Copy the value passed to the function call | Passed address to the function call |
| If we make any changes inside the function, these changes would not be reflected in other functions. | If we make any changes inside the function. These changes would be reflected in other functions. |
| The actual parameters and formal parameters are created in different memory locations. | The actual parameters and formal parameters are created in the same memory locations. |

## Truyền tham trị trong C?

* Truyền tham trị trong C là truyền cho đối số một bản sao, nghĩa là giá trị của biến đó sẽ được sao chép sang biến tham số truyền vào hàm. Và được lưu trong vùng nhớ stack
* Khi thay đổi giá trị của biến trong function thì giá trị của biến được truyền vào bên ngoài hàm sẽ k bị thay đổi giá trị

## Truyền tham chiếu trong C?

* Chính là truyền vào một bản gốc thông tin qua địa chỉ “&”. Tức là biến được sử dụng trong hàm và biến ngoài hàm truyền vào sẽ dùng chung địa chỉ.
* Khi thay đổi giá trị biến bên trong hàm thì biến được truyền vào hàm bên ngoài sẽ thay đổi theo.

What different passed by pointer vs passed by reference?

|  |  |
| --- | --- |
| **Con trỏ** | **Tham chiếu** |
| Khai báo: Int \*p=&a; int \*p; p=&a | Int &p=a; |
| Có thể chỉ định để truy cập đến các đối tượng khác nhau bằng cách thay đổi giá trị địa chỉ mà con trỏ lưu trữ. Vì vậy con trỏ linh hoạt hơn tham chiếu. | Tham chiếu chỉ truy cập được duy nhất một đối tượng đã được khởi tạo cho nó. |
| Kích thước có định (8 bytes hoặc 4 bytes phụ thuộc vào hệ điều hành) | Kích thước phụ thuộc và kiểu dữ liệu của đối tượng |
| Con trỏ có thể gán NULL trực tiếp | Tham chiếu k thể gán NULL trực tiếp. |
| Các phép toán số học khác nhau có thể thực hiện trên con trỏ | Không thể thực hiện trên các phép toán số học. |
|  | Sử dụng là tham số trong các hàm có kiểu trả về |

## **What difference between #define vs Const in C?**

|  |  |
| --- | --- |
| #define | const |
| Handled by pre-processor | Handled by compiler |
| Macro directive | Consider variable |
| Can’t type check | Cant type check |

## **What different between Maloc vs Calloc?**

**Giống nhau:** đều được sử dụng để cấp phát bộ nhớ động cho chương trình. Hàm trả về NuLL nếu không đủ vùng nhớ.

**Khác nhau:**

|  |  |
| --- | --- |
| **Malloc** | **Calloc** |
| **Nhận 1 tham số truyền vào là số byte của vùng nhớ cần cấp phát.** | **Nhận 2 tham số truyền vào là số Block và kích thước mỗi Block(byte)** |
| **Hàm trả về con trỏ trỏ tới vùng nhớ nếu cấp phát thành công. Không tự động khởi tạo bộ nhớ được phân bổ, chứa giá trị rác.** | **Hàm trả về con trỏ trỏ tới vùng nhớ được cấp phát và vùng nhớ được khởi tạo giá trị bằng 0 (auto)** |

## **What would be the output of the following code:**

#include<stdio.h>

int main()

{

int number = 100;

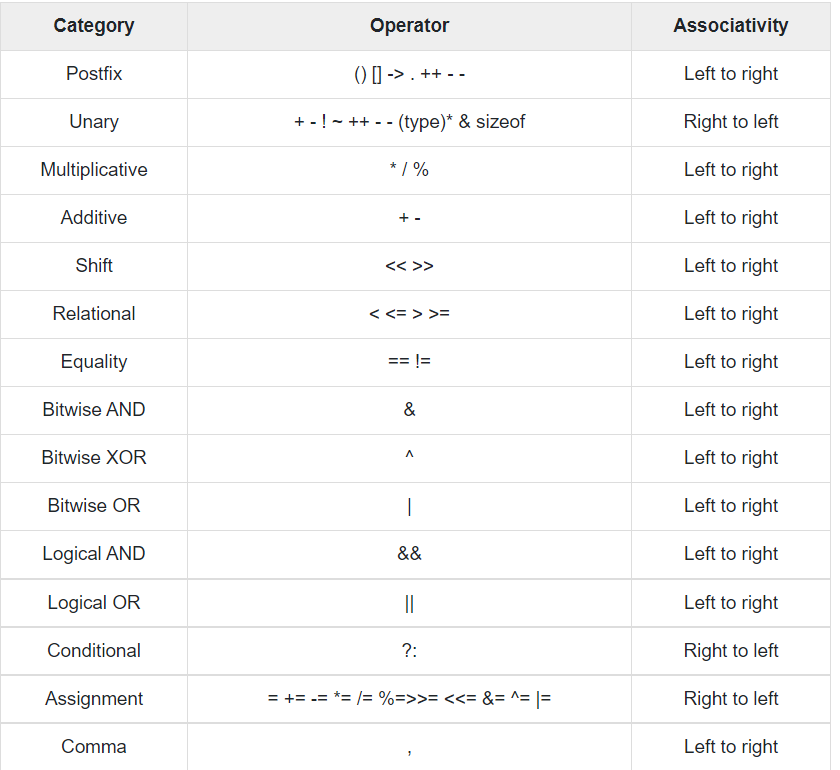
printf("%d\n%d\n%d\n",number++,number,++number);

}

// 101, 102, 102

* Allow precedence operator in C
* Perform ++number fisrt, then number++, final function printf() will execute

## Precedence operator in C:



***Postfix:*** *hậu tố*

***Prefix****: tiền tố*

## **Precedence and associativity of postfix ++ and prefix ++ are different**

* Postfix ++ is more than prefix ++,
* Associate of postfix++ is left to right, associate of prefix ++ is right to left.

***Unary:*** *là toán hạng dựa trên 1 toán tử duy nhất để tạo ra 1 giá trị mới*

*unary prefix*

*operators always associate right-to-left (sizeof ++\*p is sizeof(++(\*p))) and unary postfix operators always associate left-to-right (a[1][2]++ is ((a[1])[2])++)*

// PROGRAM 1

#include <stdio.h>

**int** main(**void**)

{

**int** arr[] = {10, 20};

**int** \*p = arr;

    ++\*p; // ++(\*p)

**printf**("arr[0] = %d, arr[1] = %d, \*p = %d",arr[0], arr[1], \*p);

**return** 0;

}

// out put 11 20 11

// PROGRAM 2

#include <stdio.h>

**int** main(**void**)

{

**int** arr[] = {10, 20};

**int** \*p = arr;

    \*p++; // \*(p++)

**printf**("arr[0] = %d, arr[1] = %d, \*p = %d",

                          arr[0], arr[1], \*p);

**return** 0;

}

// 10 20 20

// PROGRAM 3

#include <stdio.h>

**int** main(**void**)

{

**int** arr[] = {10, 20};

**int** \*p = arr;

    \*++p;//

**printf**("arr[0] = %d, arr[1] = %d, \*p = %d",

                          arr[0], arr[1], \*p);

**return** 0;

}

**// 10 20 20**

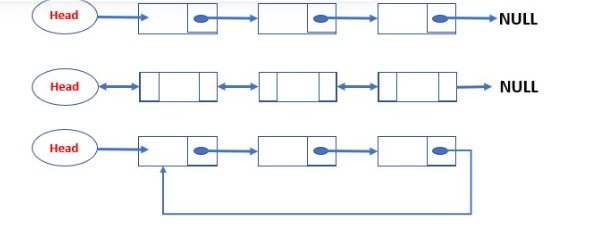
**Comma has the least precedence among all operators and should be used carefully**

## Sự khác nhau giữa array và linked list?

**Khái niệm:**

**Array:** mảng là cấu trúc dữ liệu chứa các giá trị hay biến có cùng kiểu dữ liệu. Mỗi phần tử mảng được gán một giá trị index riêng. Mảng gồm các ô nhớ được xếp liền kề nhau.

**Linkedlist**: là một danh sách cấu trúc dữ liệu tuyến tính trong đó các phần tử được liên kết với nhau nhờ các Links, mỗi phần tử sẽ bao gồm data và links (phần liên kết tới phần tử kế tiếp). Phần tử cuối cùng trỏ tới NULL.



**Cấu trúc:**

+ Array được gắn với một index cố định. Trước khi muốn truy xuất một phần tử nào đó cần biết được index của nó(từ 0 – (size -1)). Có thể truy xuất bất cứ phần tử nào có trong mảng.

+ Linkedlist: được liên kết theo thứ tự, liên kết với phần tử đứng trước và sau nên khi muốn truy xuất 1 phần tử cần truy xuất các phần tử liền kề trc nó.

**Kích thước:**

+ Array: Yêu cầu kích thước cố định khi khai báo.

+ Linkedlist: linh hoạt về kích thước.

**Bộ nhớ:**

+ Array: yêu cầu bộ nhớ trong quá trình biên dịch.

+ Linkedlist: Yêu cầu bộ nhớ trong quá trình runtime.

+ Array: các phần tử được lưu trữ liền kề nhau trong vùng nhớ stack.

+ linkedlist: lưu trữ tại các vùng nhớ khác nhau ngẫu nhiên trong Heap.

**Thời gian truy xuất:**

+ Array: có thể truy xuất trực tiếp hoặc truy xuất tuần tự từng phần tử.

+ Linkedlist: bắt buộc phải truy xuất từ phần tử đầu tiên.

**Thêm và xóa:**

+ Array: thêm và xóa phần tử trong mảng phức tạp hơn so với linkedlist.

+ Linkedlist: thay đổi liên kết của phần tử đứng trước và sau vị trí muốn thêm hay xóa.

**Tổng kết:**

Ưu điểm của linkedlist: kích thước linh hoạt, dễ dàng thêm xoá phần tử, không yêu cầu cấp phát bộ nhớ.

Ưu điểm của Array: dễ dàng truy xuất phần tử, dễ thực thi và sử dụng.

## How will you print “Hello world” without semicolon?

Int main(void)

{

If(printf(“Hello World”)) {}

}

## When should we use pointer in C?

* To get a address of a variable
* For pass by reference in C
* To implement “Linked” data structures like linker list and binary trees

## Difference call by value vs call by reference?

|  |  |
| --- | --- |
| **Call by value** | **Call by reference** |
| The value of actual parameter is copied into the formal parameters. | The address of the variable is passed into the function call |
| Can not modified the value of the actual parameter by formal parameter | The value of actual parameters can be modified by changing the formal parameter sine the address of the actual parameter passed |
| Different memory is allocated for actual parameter and formal parameter. Value actual copied into formal | Memory allocated is similar. All the operators in the function are performed on the value stored at the address of the actual parameter |
| Actual parameter is the argument used in function call |  |
| Formal parameter used in function definition |  |

## What is NULL pointer?

* NULL use to initialized the pointer doesn’t point to any address

## Memory leak? Why should be avoid?

* Memory leak occurs when programmer create memory in Heap and forget to delete it.
* If memory leak occurs lead to program never terminate (k bh ket thuc)
* Rò rỉ bộ nhớ trong C xảy ra khi chúng ta phân bổ 1 vùng nhớ trong heap bằng cách sử dụng hàm malloc hoặc calloc nhưng quên free sau khi sử dụng xong. Dần dần chương trình chạy mà dữ liệu trước đó k được free đi dẫn đến tràn bộ nhớ Heap và dẫn tới lỗi Memory leak.

## What different between i++ vs ++i?

|  |  |
| --- | --- |
| **i++** | **++i** |
| Return old value, then increments i | Increments the value and returns new value |
| Precedence left-to-right | Right-to-left |
| Precedence higher than | Lower than |

## How to write own size of operator?

#define my\_sizeof(type) (char\*)(&type+1)-(char\*)(&type)

## Different Compiler time vs Runtime?

|  |  |
| --- | --- |
| **Compile time** | **Run time** |
| Là thời gian chuyển đỗi Source code sang executable code | Là thời gian thực thi executable code, xảy ra sau quá trình compile |
| Phát hiện các lỗi về syntax, lỗi cấu trúc, ngữ nghĩa. | Các lỗi như chia cho 0, căn bậc 2 của 1 số âm |
| Phát hiện lỗi bởi compiler | Các lỗi k do trình biên dịch tạo ra và kết quả k thể đoán trước tại thời điểm thực thi. |
| K cho thực thi nếu phát hiện ra lỗi | Vẫn thực thi sau đó với phát hiện ra lỗi |

## Can a variable be both const and volatile?

* Yes, the const mean cannot be assigned a new value, but can be change by other code or pointer .

#include <stdio.h>

**int** main(**void**)

{

**const** **volatile** **int** local = 10;

**int**\* ptr = (**int**\*)&local;

**printf**("Initial value of local : %d \n", local);

    \*ptr = 100;

**printf**("Modified value of local: %d \n", local);

**return** 0;

}

## Ưu nhược điểm của Array trong C?

* Ưu điểm:
  + Dễ sử dụng
  + Truy cập đến các phần tử trong mảng nhanh.
* Nhược điểm:
  + Kích thước cố định, cần xác định kích thước trước khi chạy ctrinh( được cấp phát khi biên dịch, trong khi đó mảng động được cấp phát khi chạy ctrinh). Kích thước đôi khi là k thể xác định trước nên khá bất tiện. Khai báo thừa gây lãng phí bộ nhớ, khai báo thiếu thì lại k đủ dùng.
  + Các byte được cấp phát liên tiếp nhau trong bộ nhớ: trong trường hợp các bytes trong bộ nhớ bị phân mảnh ctrinh sẽ dẫn đến lỗi k đủ vùng nhớ ltuc.
  + Việc chèn xóa mảng mất nhiều thời gian.

## EX1: A là một số 10 bit

Thanh ghi B chứa 8 bit cao của A

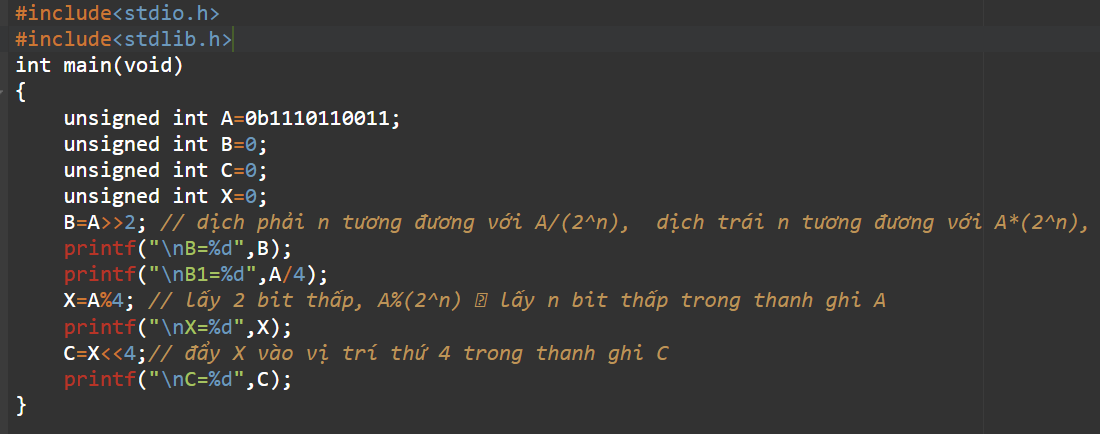
2 bit 4 và 5 của thanh ghi C chứa 2 bit thấp của A

Tại sao:

B=A/4

C=(A%4)<<4

Code:

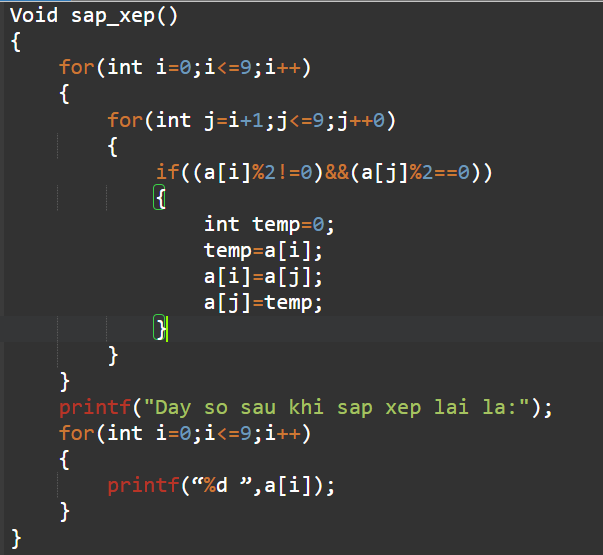


## \_\_Weak keyword in C?

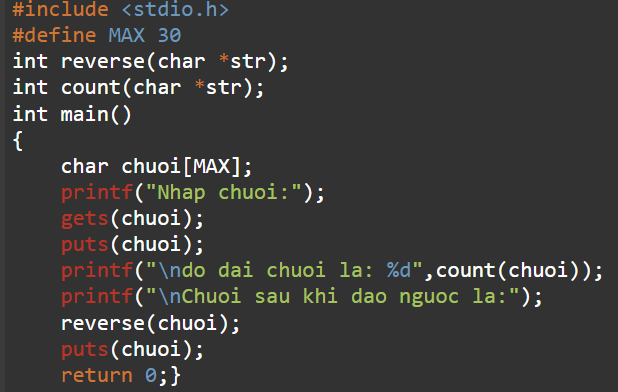
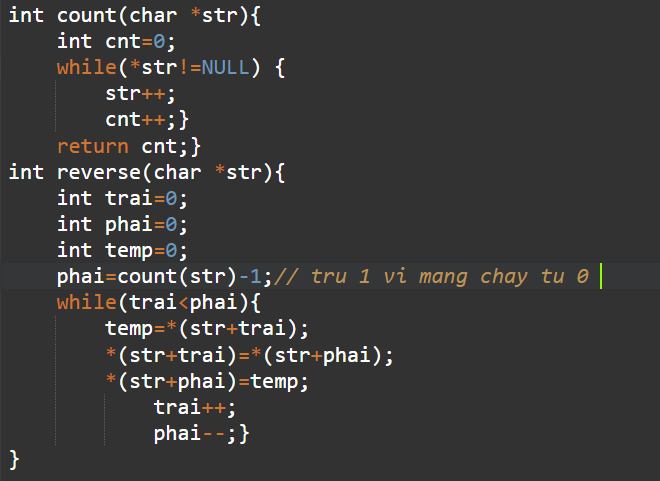
* Provide a function empty for custemer use

## Sắp xếp dãy số 10 chữ số chẵn trước lẻ sau?

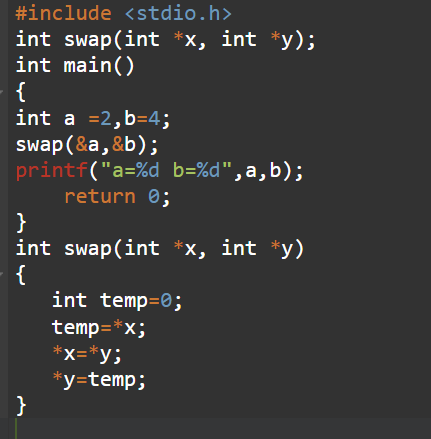
Code:



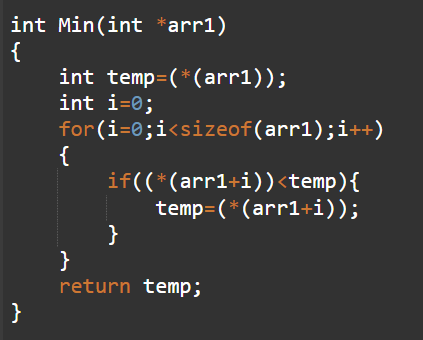
## Đếm số phần tử và đảo ngược chuỗi string?

Code:

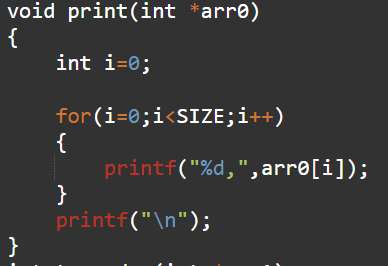
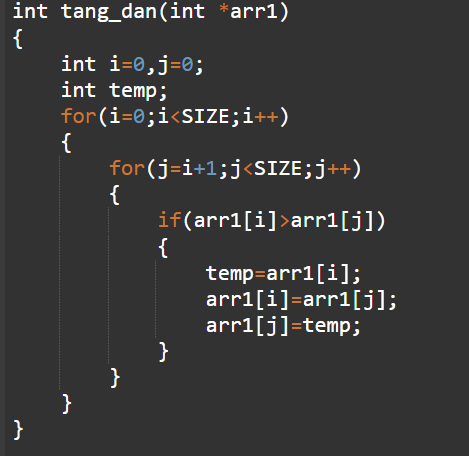
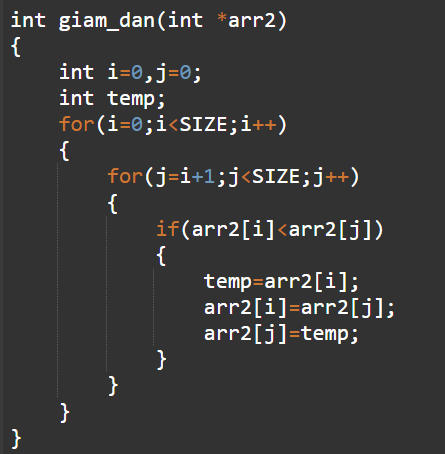
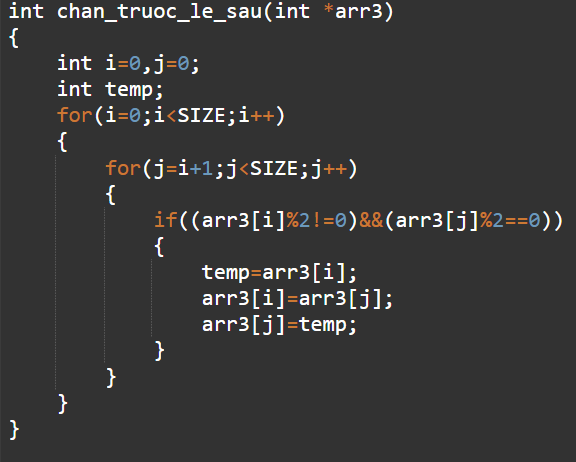
## Swap 2 số?

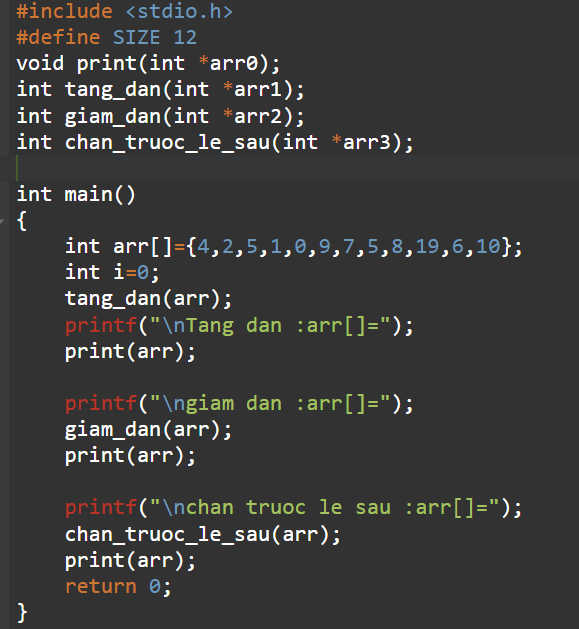


## **Tim Min Max trong mang?**



## Sắp xếp mảng theo thứ tự tăng dần, giảm dần, chẵn trước lẻ sau?

**Code:**



# **MICROCONTROLER QUESTIONS**

## What main components in Microcontroller?

+ inputs/outputs ports

+ Core (microprocessor core)

+ Read only memory (ROM)

+ counters and timer

+ Random access memory

+ ADC, DAC , Serial interface

+ Interrupt controls

+ PWM

+ Clock

+Bus interface

RAM

ROM

Clock

I/O

PERIPHERAL

CORE

ADC/DAC

SERIAL INTERFACE

BUS INTERFACE

## Types of memory use in microcontroller?

* **Flash memory (Non-volatile):** Holds program code and some data, as know code memory, code memory often larger than RAM memory. Flash can’t be overwritten during program execution and can be overwritten only once the device is reprogrammed. Flash only read during program execution
* **RAM (Random access memory - volatile**): RAM holds different types of program data such as temporary variables, global variables, stack, heap sections. RAM has lower latency (do tre) than Flash memory. Cost RAM higher than Flash memory. Size of RAM lesser than FLASH memory.
* **CPU Register (Volatile):** CPU register holds operands, instructions addresses, run-time state of the program. 2 types register general purpose register and special function register. General purpose Register hold values of operands.

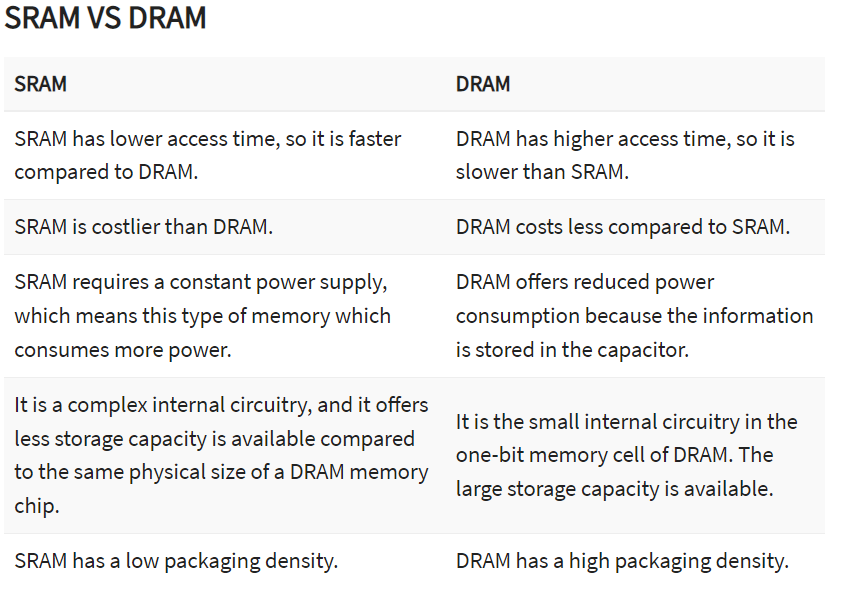
Special function register holds program counter, main stack pointer, instruction set

## What is RAM and types?

* RAM use to store the programs and data being use be read, written and erased any number of times. Volatile memory.

+ **SRAM (Static RAM):** which stores a bit of data using the state of a six transistors memory cell. RAM is mostly used as a cache memory for the CPU

+ **DRAM (Dynamic RAM)**: Which stores a bit data using a pair of transistor and capacitor which constitute a DRAM memory cell.



## What is ROM and types?

* **ROM (Read only memory):** is a typesof memorywhere the data has been prerecorded(nap truoc).non-volatile.

+ **Programmable ROM(PROM)**: where the data is written after the memory chip has been created.

+ **Erasable Programmable ROM (EPROM):** Can be erased by exposing (txuc) it to high-intensity UV light.

+ **Electrically Erasable programmable ROM(EEPROM):** can be electrically erased using field electron emission.

+ **Mask ROM:** in which the data is written during the manufacturing of the memory chip.

## Different RAM and ROM?

| **Difference** | **RAM** | **ROM** |
| --- | --- | --- |
| Data retention | RAM is a volatile memory which could store the data as long as the power is supplied. | ROM is a non-volatile memory which could retain the data even when power is turned off. |
| Working type | Data stored in RAM can be retrieved(Truy xuat) and altered.(thay doi) | Data stored in ROM can only be read. |
| Use | Used to store the data that has to be currently processed by CPU temporarily. | It stores the instructions required during bootstrap of the computer. |
| Speed | It is a high-speed memory. | It is much slower than the RAM. |
| CPU Interaction | The CPU can access the data stored on it. | The CPU can not access the data stored on it unless the data is stored in RAM. |
| Size and Capacity | Small size with less capacity. | Large size with higher capacity. |
| Used as/in | CPU Cache, Primary(chinh) memory. | Firmware, Micro-controllers |
| Accessibility | The data stored is easily accessible | The data stored is not as easily accessible as in RAM |
| Cost | Costlier | cheaper than RAM. |
| Storage | A RAM chip can store only a few megabytes of data. | A ROM chip can store multiple gigabytes (GB) of data |

## Different Volatile memory vs Non-volatile memory?

* **Volatile memory**: it is also call as temporary memory. When the system is turned off the data within the volatile memory is deleted automatically. 2 types are **RAM vs Cache** memory.(Read-write)
* **Non**-volatile memory: data or information store within the Non-volatile memory not lost within the memory even power is shut-down. (ROM is Non-volatile memory)

|  | **Volatile Memory** | **Non-Volatile Memory** |
| --- | --- | --- |
| 1. | Volatile memory is the type of memory in which data is lost as it is powered-off. | Non-volatile memory is the type of memory in which data remains stored even if it is powered-off. |
| 2. | Contents of Volatile memory are stored temporarily. | Contents of Non-volatile memory are stored permanently. (vinh vien) |
| 3. | It is faster than non-volatile memory. | It is slower than volatile memory. |
| 4. | **RAM(Random Access Memory)** is an example of volatile memory. | **ROM(Read Only Memory)** is an example of non-volatile memory. |
| 5. | In volatile memory, data can be easily transferred in comparison to non-volatile memory. | In non-volatile memory, data can not be easily transferred in comparison to volatile memory. |
| 6. | In Volatile memory, process can read and write. | In Non-volatile memory, process can only read. |
| 7. | Volatile memory generally has less storage capacity. | Non-volatile memory generally has more storage capacity than volatile memory. |
| 8. | In volatile memory, the program’s data are stored which are currently in process by the CPU. | In non-volatile memory, any kind of data which has to be saved permanently are stored. |
| 9. | Volatile memory is more costly per unit size. | Non-volatile memory is less costly per unit size. |
| 10. | Volatile memory has a huge impact on the system’s performance. | Non-volatile memory has a huge impact on a system’s storage capacity. |
| 11. | In volatile memory, processor has direct access to data. | In non-volatile memory, processor has no direct access to data. |
| 12. | Volatile memory chips are generally kept on the memory slot. | Non-volatile memory chips are embedded on the motherboard. |
| 13 | **Advantages-**   * Fast speed * Low power consumption * Better system performance as it increases speed | **Advantages-**   * More reliable * Stores data permanently * Inexpensive memory * Helps in booting of operating system |
| 14 | **Disadvantages-**   * Expensive * Limited storage space * Stores data temporarily | **Disadvantages-**   * Slow speed * Can only read data |

## What is Cache and function of Cache?

* Cache memory is faster, they can be accessed very faster.
* Cache memory is smaller, a large amount of data cannot be stored

Whenever CPU needs any data it searches for corresponding(tuong ung) data in the cache (fast process). If data not found in the cache CPU it will be search that data in primary memory (slower process) and load it into the cache.

* Cache have 3 levels L1, L2, L3.

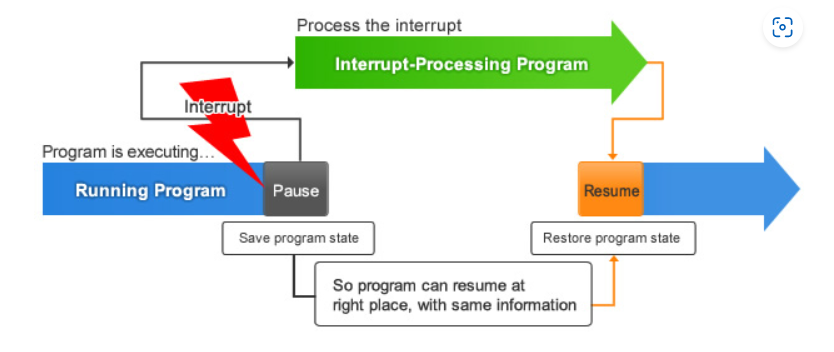
## Cache vs RAM?

|  |  |
| --- | --- |
| **RAM** | **Cache** |
| RAM is larger in size compared to cache. Memory ranges from 1MB to 16GB | The cache is smaller in size. Memory ranges from 2KB to a few MB generally. |
| It stores data that is currently(hien tai) processed by the processor. | It holds frequently(thuong xuyen) accessed data. |
| OS interacts with secondary memory to get data to be stored in Primary Memory or RAM | OS interacts with primary memory to get data to be stored in Cache. |
| It is ensured that data in RAM are loaded before access to the CPU. This eliminates RAM miss never. | CPU searches for data in Cache, if not found cache miss occur. |

## Different microprocessor vs microcontroller?

|  |  |
| --- | --- |
| **Microprocessor** | **Microcontroller** |
| Is the heart if computer system | Is the heart if an embedded system |
| It only a processor, so memory and I/O components need to connected external | Has processor along with internal memory and peripheral . |
| Use in personal computers or desktop | Use in a washing machine, MP3, embedded systems… |
| It is a dependent unit | The microcontroller is used to perform a particular tasks. |
| Its power consumption is high | Its power consumption is low. |
| It requires a combination of timers, controllers memory chips. | It contains CPU, RAM, ROM, Registers, Timer and input/output ports. |
| Its size is larger. | Its size is smaller. |
| Its processing power is higher. | Its microprocessors processing power is lower than microprocessor. |

## What is interrupt in microcontroller?



**B1:** Thực hiện xong câu lệnh đang thực hiện

**B2:** save program state, lưu địa chỉ câu lệnh tiếp theo sẽ thực hiện, lưu trạng thái năng lượng đang hoạt động vào vùng nhớ stack.

**B3:** Xóa bit cho phép ngắt toàn cục trong thanh ghi status, đưa VDK về chế độ hoạt động bthg (active mode) nếu đang hoạt động ở chế độ tiết kiện năng lượng.

**B4:** Vi điều khiển thực thi chương trình phục vụ ngắt (ISR) bằng cách nạp địa chỉ câu lệnh đầu tiên của chương trình phục vụ ngắt vào thanh ghi PC

**B5:** quá trình unstacking thực thi. Nạp lại giá trị thanh ghi PC đã lưu (địa chỉ câu lệnh tiếp theo ở B1)

Một số ngắt phổ biến trên VDK:

+ Ngắt ngoài: khi có sự thay đổi sườn tín hiệu (edge) sườn lên, sườn xuống

+ Ngắt UART: thường là ngắt nhận, ngắt xảy ra khi buffer nhận đủ 1 byte dữ liệu

+ Ngắt ADC: Thường sử dụng khi hoàn thành việc chuyển đổi ADC

+ Ngắt Timer: Thường sử dụng khi tràn thanh ghi đếm, hoặc khi giá trị đếm bằng với thanh ghi so sánh.

## Timer/counter in microcontroller?

Timer ứng dụng: xác định chính xác một khoảng thời gian, đo- đếm xung đầu vào, điều khiển sóng đầu ra, băm xung(PWM)...VD: thay đổi trạng thái LED sau bao nhiêu giây. Dùng PWM điều khiển tốc độ động cơ, độ sáng bóng đèn.

Thành phần chính của timer chính là bộ đếm – **Counter**, với các ngưỡng được thiết lập bởi thanh ghi **auto reload** (ARR).

* Các chế độ hoạt động:

+ Upcounting mode (chế độ đếm lên).

+ Downcounting mode (Chế độ đếm xuống).

+ Center – Aligned mode (Chế độ đếm lên và xuống).

## Chuẩn CMSIS là gì?

* **CMSIS** is Common Microcontroller Software Interface Standard mục đích chính để tương thích phần mềm cũng như sử dụng lại mã nguồn trở nên đơn giản và dễ dàng hơn.
* Việc có thiết bị mới người lập trình cần phải viết lại phần mềm điều khiển để tương thích với thiết bị mới, điều này làm tăng thời gian phát triển sản phẩm cũng như giá thành của dự án.
* Ví dụ ứng dụng cần được nâng cấp từ STM32F1 lên STM32F4 thì thông thường chúng ta cần phát triển lại chương trình lại từ đầu do đó CMSIS là cần thiết.
* Do đó chuẩn CMSIS ra đời, do ARM cung cấp. Các nhà sản xuấ chip đã cung cấp 1 khung giao diện phần mềm chuẩn miễn phí cho việc điều khiển và truy cập thiết bị.

## Chuẩn giao tiếp UART?

* Là 1 chuẩn giao tiếp truyền thông nối tiếp (tức là chuyển từng bit dữ liệu qua một đường dây )
* Truyền thông nối tiếp có tốc độ truyền thấp hơn song song nhưng lại dùng ít dây hơn=> được sử dụng nhiều hơn.

+ **Truyền đồng bộ:** Truyền dữ liệu số lượng lớn trong cấu trúc đóng khung tại một thời điểm. Truyền nhận theo xung clock

+ **Truyền không đồng bộ:** Truyền theo từng byte trong cấu trúc đóng khung tại một thời điểm. Truyền nhận không theo xung clock

UART thuộc loại truyền không đồng bộ (Asynschronus) (Tx, Rx, GND).

Cấu trúc khung truyền thường bao gồm:

+ Start Bit. Luôn là 1, để bắt đầu truyền dữ liệu kéo xuống 0

+ Gói dữ liệu (5-9 bits thường là 8)

+ Parity bit

+ Stop bit ( 1 hoặc 2 bit)

* Tốc độ truyền dữ liệu – Baudrate (số bit truyền đi trong mỗi giây)

## Chuẩn giao tiếp SPI?

**SPI** is Serial peripheral interface: là một chuẩn truyền thông nối tiếp, đồng bộ, khác với UART là không đồng bộ. SPI hoạt động dựa trên cơ chế Master-slave và có thể có nhiều Slave trên đường bus giao tiếp.

Sơ đồ chân:

+ **MOSI:** Master Out – Slave in, là chân truyền dữ liệu của Master và nhận dữ liệu của Slave.

+ **MISO:** Master In – Slave Out, là chân truyền dữ liệu từ Slave và Master nhận dữ liệu.

+ **SCLK:** chân cung cấp Clock cho Master.

+ **SS/CS :** Chip select, chân chọn chip, hoạt động tích cực mức thấp.

Phần cứng SPI bao gồm một bộ thanh ghi dịch 8-bits để chứa dữ liệu truyền/ nhận. Bên Master có 1 bộ Clock Genarator để cấp xung đồng bộ.

Mỗi chu kỳ xung mà Master cấp cho Slave , một bit dữ liệu sẽ được đẩy đi trên thanh ghi dịch.

Các chế độ hoạt động:

+ CPOL – Clock polarity, xác định trạng thái Clock ở trạng thái nghỉ.

+ CPHA- clock Phase , xác định sườn xung Rising / failling để truyền data.

Ứng dụng: các loại bộ nhớ SD card, EEPROM, FLASH.. các loại cảm biến (Nhiệt độ áp suất..), các bộ chuyển đổi ADC, DAC...

## Chuẩn giao tiếp I2C?

* I2C là chuẩn giao tiếp nối tiếp đồng bộ thường được sử dụng để giao tiếp giữa các IC trong cùng bo mạch.
* I2C is Inter-integrated Circuit, là một chuẩn truyền thông nối tiếp, đồng bộ (dữ liệu truyền nhận được đồng bộ theo xung Clock, 1 chu kỳ xung nhịp sẽ cho phép truyền nhận 1 bit dữ liệu )
* Hoạt động theo cơ chế multi Master-Slave và có thể có nhiều Master và Slave trên đường Bus giao tiếp.
* 2 Chân:

+ **SDA:** Serial Data, chân truyền dữ liệu Master – Slave

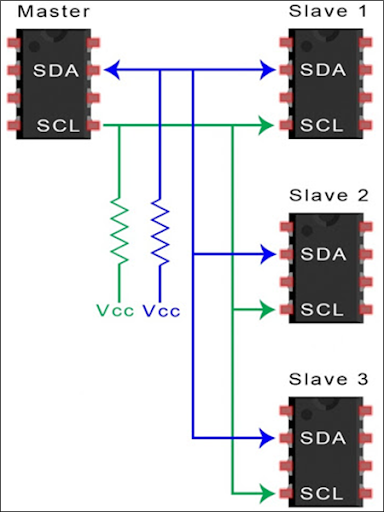
+ **SCL:** serial clock là chân cấp clock từ các master sang Slve để đồng bộ dữ liệu.

**Đặc điểm của giao thức I2C:**

Hoạt động theo cơ chế half-duplex. Với việc có nhiều master cũng như nhiều slave trên một đường Bus data nên việc quản lý master nào sẽ giao tiếp với slave nào sẽ được quy định bằng phương pháp đánh địa chỉ (7 bit địa chỉ)

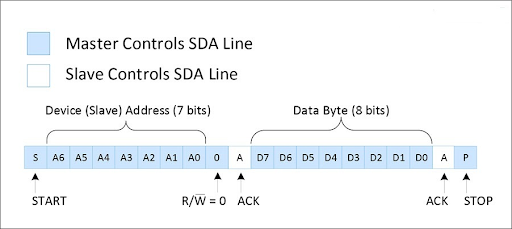
Mỗi Slave sẽ được đánh 1 địa chỉ cố định duy nhất và phân biệt với các slave khác. Khi master nào muốn giao tiếp với Slave nào, nó sẽ gửi địa chỉ trên đường SDA, và Slave nào có địa chỉ tương ứng sẽ gửi phản hồi về master để bắt đầu giao tiếp.

2 đường bus thường có điện trở pull up để kéo đường truyền mặc định lên mức cao.



**Cơ chế truyền nhận dữ liệu:**

Khung truyền



Start bit SDA được kéo từ mức cao về mức thấp, SCL luôn giữ ở mức cao

Stop bit SDA được kéo từ mức thấp lên mức cao, SCL luôn giữ ở mức cao

Sau start bit, Master sẽ gửi 7 bit địa chỉ cùng với đó là bit Read/write để xác nhận Masternafy muốn truyền hay nhận data. Nếu Slave nào trùng địa chỉ với địa chỉ Master truyền sẽ phản hồi về Master 1 bit ACK. Các Slave còn lại sẽ gửi bit NACK( mặc định mức cao).

Sau đó Master sẽ gửi các Byte data

Sau khi truyền xong data Master sẽ gửi Stop Condition.

**Ứng dụng:**

+ Giao tiếp giữa Vi điều khiển và các mảnh cảm biến

+ Giao tiếp hiển thị (LCD..)

+ Giao tiếp 1 số thiết bị IoT..

# **SOFTWARE TESTING**

What are level of testing?

4 levels:

+ Unit testing

+ Integration testing

+ System testing

+ Acceptence testing

What are the phases of the Software Development life cycle?(SDLC)

+ **Requirement:** Nhận yêu cầu từ khách hàng

+ **Analysis:** phân tích yêu cầu và convert toàn bộ yêu cầu của khách hàng sang dạng documents.

Đưa lại tài liệu này cho bên khách hàng xem bản mô tả requirment này đã đúng với yêu cầu của họ chưa .

+ **Design:** bao gồm 2 phase:

- Low level design Documents: thực hiện bởi senior developer nơi họ sẽ thiết kế cụ thể từng chức năng của module.

- Hight level design document: cả tester and developer đều thực hiện ở phase này. Thiết kế các architecture khác nhau cần được phát triển.

+ **Coding**: tất cả những yêu cầu sẽ được chuyển sang bằng việc coding

+ **Testing:** Đảm bảo phần mềm k còn lỗi hoặc lỗi ít nhất có thể để đảm bảo chất lượng sản phẩm.

-Static testing: check các lỗi trong ứng dụng mà k cần execute code. Tránh lỗi ở giai đoạn đầu của quá trình phát triển. Dễ dàng phát hiện và xử lý.

-Dynamic testing: test trên môi trường và thiết bị tương tự như môi trường thực tế kiểm tra xem hoạt động có đúng như yêu cầu...

+ **Maintance:** bảo trì bảo dưỡng, nâng cấp trong tương lai

What is equivalence partitioning?(phân vùng tương đương)

* Là phương pháp chia điều kiện đầu vào thành những vùng tương đương nhau. Tất cả các giá trị trong một vùng tương đương sẽ cho kết quả đầu ra giống nhau. Vì vậy chỉ cần test 1 giá trị đại diện trong vùng tương đương đó. Thuộc black box testing
* Ưu điểm: giảm số lượng test case và thời gian test do chỉ cần test phần tử đại diện
* Nhược điểm: kp bất kỳ bài toán nào cũng có thể áp dụng kỹ thuật này. Có thể sót lỗi ở biên nếu chỉ test ở khoảng giữa của miền tương đương

What is Boundry Value analysis? (phân vùng giá trị biên)

* Phân vùng giá trị biên là quá trình kiểm tra các giá trị ranh giới giữa các phân vùng của giá trị đầu vào(gtri lớn nhất, nhỏ nhất)
* Để tiến hành kiểm thử giá trị biên cần phân vùng tương đương trước đó.

What is Unit testing?

* Là kiểm tra đơn vị nó đóng vai trò là một công cụ kiểm thử các thành phần riêng lẻ của một phần mềm như function, classs.. Thường Unit test sẽ được thực hiện trong khi chương trình đang được phát triển.
* Mục đích là khoanh vùng và kiểm tra tính chính xác của các vùng code.

What is integration test?

* Là kiểm thử tích hợp. Mỗi module phần mềm riêng biệt được kết hợp lại và kiểm thử theo nhóm.
* Xảy ra sau kiểm thử đơn vị (Unit test )và trước System test
* Nhận các module đầu vào đã được kiểm thử đơn vị.
* Đảm bảo tính hợp nhất của phần mềm khi ráp các module lại với nhau.
* Khi tích hợp hệ thống các module có thể không tương thích với cấu hình chung của hệ thống.

What is System testing?

* Là 1 cách theo dõi và đánh giá hành vi của sản phẩm hoặc hệ thống phần mềm hoàn chỉnh đã được tích hợp đầy đủ, dựa vào các đặc tả và các yêu cầu chức năng đã được xác định trước.
* Là một loại kiểm thử hộp đen
* Đánh giá chức năng của hệ thống hoàn chỉnh theo yêu cầu chức năng đã được định trước.

What is Acceptance testing?

* Là một kiểm thử nhằm xác định hệ thống phần mềm có đạt yêu cầu kỹ thuật hay không. Việc kiểm thử chấp nhận nhằm xác định hệ thống có đạt yêu cầu lẫn tiêu chí của khách hàng hay k
* Kiểm chứng trực tiếp
* Có cái nhìn tổng quan nhất về kết quả hệ thống đạt được

What is Black-box testing?

* Là phương pháp kiểm thử phần mềm mà không biết được cấu tạo bên trong phần mềm gồm những gì, không làm việc trực tiếp với code và đọc code. Kiểm thử chủ yếu chức năng hệ phần mềm
* Phương pháp này có thể tìm ra các loại lỗi như sau:

+ chức năng không chính xác hoặc thiếu

+ hành vi hoặc hiệu suất lỗi

+ lỗi giao diện..

* Thử nghiệm được áp dụng ở cấp độ cao như: System testing , acceptance testing
* Không cần hiểu biết về lập trình
* Dựa trên tài liệu yêu cầu kỹ thuật

What is White-box testing?

* Là phương pháp kiểm thử phần mềm sử dụng để kiểm thử phần mềm mà yêu cầu phải biết cấu trúc bên trong của chương trình (code).
* Kiểm thử được áp dụng ở mức độ thấp như Unit testing, integration testing
* Yêu cầu hiểu biết nhất định về lập trình
* Hiểu rõ cấu trúc bên trong chức năng được thực hiện như nào
* Dựa trên tài liệu thiết kế chi tiết

What is Functional Testing?

* Là loại kiểm thử tập trung và các yêu cầu chức năng của sản phẩm, phần mềm. Chủ yếu liên quan đến các kỹ thuật kiểm thử hộp đen.
* Nhờ có functional testing mà bạn có thể tránh được những kiểm thử dư thừa các chức năng k cần thiết.

What is Non-functional testing?

* Kiểm tra các khía cạnh phi chức năng (hiệu suất, khả năng sử dụng, độ tin cậy..) của ứng dụng, phần mềm.

What is test case?

* Là các đầu vào input, điều kiện kiểm thử, quy trình kiểm thử và kết quả mong muốn với một mục tiêu kiểm thử cụ thể. Đó là kiểm tra các tính năng của phần mềm trước khi đến với người dùng cuối.
* Test case được viết ra với mục đích: cải thiện chất lượng phần mềm, đảm bảo phạm vi kiểm thử, giảm chi phí bảo hành và hỗ trợ phần mềm, xác định phần mềm có đáp ứng nhu cầu của người dùng cuối hay k
* Các bước viết test case:

+ Đặt tiêu đề

+ giới thiệu về test case. Phân tích tính năng nào sẽ đk test, môi trường kiểm thử, dữ liệu kiểm thử.

+ đặt ra các giả thiết được coi là điều kiện tiên quyết và cần thiết cho quá trình kiểm thử.

+ trình bày các bước kiểm thử chính, rõ ràng

+ Đưa ra kết quả mong muốn từ đo tester xác định kiểm thử là pass hay fail.

What is V model?

* Là dạng SDLC có tính chặt chẽ, trong đó có một giai đoạn kiểm thử chạy song song với mỗi giai đoạn phát triển. Mô hình chữ V là một phần mở rộng của mô hình thác nước (Waterfall).

Acceptane testing

Requirent

**Validation**

**Verification**

System desing

System testing

High level design

Integration testing

Low level design

ign

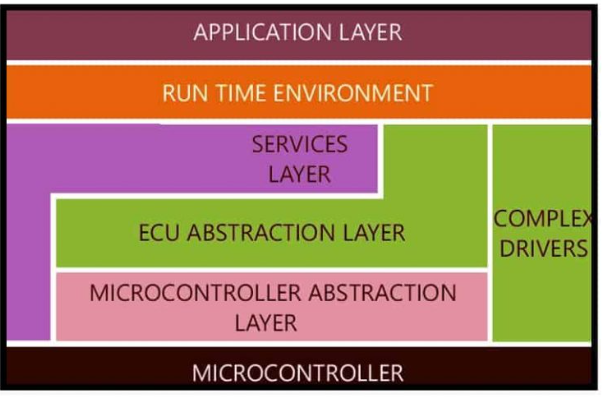
Unit testing

**CODING**

AUTOSAR (AUTomotive Open System ARchitecture)

Autosar là gì?

* Tiêu chuẩn hóa kiến trúc phần mềm cho ngành công nghiệp ô tô
* Sở dĩ họ phải thiết lập ra những tiêu chuẩn chung vì sự phức tạp của phần mềm điện tử trên xe hơi ngày càng tăng cao.
* Mô hình autosar:



+ Tầng 1: Microcontroller

+ Tầng 2: BSW (Basic soft ware)

+ Microcontroller Abstraction layer

+ ECU Abstraction layer

+ Complex Drivers

+ Services Layer

+ Tầng 3: Run time Enviroment (RTE)

+ Tầng 4: Application layer