1. **TÌM HIỂU VỀ LẬP TRÌNH C CƠ BẢN.**
2. Ngôn ngữ C là gì, ưu và nhược điểm, phạm vi ứng dụng?

* C là một **ngôn ngữ lập trình cấp trung**, ngôn ngữ lập trình C được phát triển bởi Dennis M. Ritchie để phát triển hệ điều hành UNIX tại Bell Labs. C được thực thi lần đầu tiên trên máy tính DEC PDP-11 vào năm 1972.
* Ưu điểm
* Là ngôn ngữ lập trình bậc trung
* Là ngôn ngữ có cấu trúc
* Là một ngôn ngữ mạnh và mềm dẻo
* C tạo ra các chương trình hiệu quả
* Là một ngôn ngữ khả chuyển
* C có kích thước nhỏ
* C định kiểu không cứng nhắc
* Nhược điểm
* C không hỗ trợ về khái niệm OOP ( kế thừa, đa hình , đóng gói, trừu tượng, ẩn dữ liệu)
* Lỗi không được phát hiện theo mỗi dòng mã
* Không thể xác định một biến có cùng tên trong C
* C không có hàm tạo và hàm hủy
* C có mức độ trừu tượng thấp

1. Biến, hằng, toán tử, macro, kiểu dữ liệu

* Biến (variable): dùng để lưu dữ liệu tạm thời trong quá trình chương trình hoạt động
* Khai báo: mọi biến cần phải khai báo trước khi sử dụng ( data\_type variable\_name; )
* Lấy địa chỉ của biến: & variable\_name
* Hằng: là đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán
* Hằng dấu phẩy động (float và double)
* Hằng int
* Hằng long
* Hằng int hệ 8
* Hằng nguyên hệ 16
* Hằng ký tự
* Hằng xâu ký tự
* Macro:
* là một tên biến bất kỳ (do lập trình viên đặt tên) trỏ tới một khối lệnh thực hiện một chức năng nào đó
* Trong quá trình tiền xử lý, các macro được sử dụng trong chương trình được thay thế bởi các khối câu lệnh tương ứng
* Định nghĩa macro bằng lệnh #define
* Kiểu dữ liệu
* Kiểu int: là kiểu số nguyên (4byte)
* Kiểu float(4 byte) và double(8 byte): kiểu số thực , dùng để chứa những số có dấu phẩy động
* Kiểu char(1 byte): dùng để lưu trữ một ký tự bất kỳ ở định dạng chuỗi
* Kiểu void: có ý nghĩa không là gì cả, thường được dùng trong những function không có giá trị trả về
* Kiểu short(2 byte) và long(8 byte): nếu bạn chắc chắn số nguyên lưu trữ rất nhỏ thì hãy sd từ khóa short, còn rất lớn thì dùng long.

1. Các cấu trúc điều kiện, vòng lặp.

* Cấu trúc rẽ nhánh
* If ( điều kiện){ statements}
* If(biểu thức điều kiện){

Câu lệnh }

else { câu lệnh }

* Switch ( biểu thức){

Case giá\_trị\_1: câu lệnh 1; [break];

Case giá\_trị\_2: câu lệnh 2; [break];

……………………………………………..

[default: câu lệnh n+1; [break];] }

* Vòng lặp
* for (lệnh khởi tạo; lệnh điều kiện; lệnh thay đổi biến điều khiển){

statements; }

* while ( biểu thức điều kiện){ statements; }
* do{

statements; } while (biểu thức điều kiện);

1. Cấu trúc dữ liệu tự định nghĩa: Struct, union

* Kiểu DL cấu trúc : struct
* Định nghĩa:
* Cách 1: không sử dụng từ khóa typedef:
* Struct tên\_kiểu{

Khai báo các trường; };

* Cách 2: sử dụng từ khóa typedef:
* typedef struct {

khai báo các trường; } tên\_kiểu;

* khai báo:
* cách 1: khai báo sau khi định nghĩa kiểu struct
* [struct] tên\_kiểu v;

[struct] tên\_ kiểu v1, v2;

* cách 2: kết hợp khai báo và định nghĩa kiểu
* struct tên\_kiểu{

khai báo các trường;} v1,v2;

* Truy cập vào các trường
* Cách 1: dùng biến thông thường sử dụng cú pháp:
* “tên\_biến.tên\_trường”
* Cách 2: dùng biến con trỏ, sử dụng cú pháp:
* “tên\_biến -> tên\_trường”
* Gán giá trị:
* Hai biến cùng một kiểu struct có thể được gán cho nhau. Việc gán này sẽ thay cho việc gán lần lượt tất cả các trường của hai biến này cho nhau.
* Kiểu dữ liệu cấu trúc: union
* Union là gì. Cũng như cấu trúc , union gồm nhiều thành phần, nhưng chúng khác nhau ở chỗ: các thành phần của cấu trúc có những vùng nhớ khác nhau, còn các thành phần của union được cấp phát một vùng nhớ chung. Độ dài của union bằng độ dài của thành phần lớn nhất.
* Về ý nghĩa sử dụng thì tương tự như cấu trúc struct. Tuy nhiên ở mỗi thời điểm trong chương trình thì chỉ dùng đến một trường trong số tất cả các trường, vì kích thước lưu trữ chỉ đủ để lưu trữ cho một trường.

1. Con trỏ, mảng.

* Con trỏ
* Trước khi nhắc về con trỏ thì ta phải nói đến địa chỉ. Địa chỉ liên quan đến một biến ta đã có khái niệm: tên biến, kiểu biến, giá trị của biến.
* Con trỏ là đối tượng dữ liệu mà giá trị của nó địa chỉ các đối tượng khác(có thể là chính nó) trong bộ nhớ.
* Khai báo biến con trỏ:
* kiểu\_dữ\_liệu\*var;
* kiểu\_dữ\_liệu\*var1, \*var2;
* khai báo hằng con trỏ
* const kiểu\_dữ\_liệu\*cons1=<exp>;
* kiểu\_dữ\_liệu\*const cons2=<exp>;
* const T\*const cons3 = <exp>;
* Phép gán: p = <bt>;
* Truy cập vào đối tượng được trỏ:
* Khi một con trỏ p đang trỏ vào một đối tượng A, ta có thể truy nhập vào A thông qua một trong các toán tử truy nhập. Trong C có nhiều toán tử truy nhập khác nhau, thường phụ thuộc vào kiểu DL của đối tượng A.
* Toán tử \*: áp dụng khi A là đối tượng DL thuộc một trong các kiểu DL cơ bản (kí tự, số nguyên, số thực). Cú pháp: “\*p”.
* Toán tử []: áp dụng khi A là dữ liệu kiểu mảng. Cú pháp: p[N], với N là một biểu thức nhận giá trị nguyên.
* Toán tử →: áp dụng khi A là DL kiểu struct. Cú pháp: p→fn, với fn là tên trường trong cấu trúc struct mà ta muốn truy nhập.
* Phép tăng giảm địa chỉ
* Đó là các phép toán (+,-, ++, --) trên biến con trỏ
* Cấp phát động bộ nhớ
* Thư viện <stdlib.h> cung cấp một số hàm để quản lý vùng nhớ động:
* Hàm malloc(): có định dạng hàm:

Void\*malloc(int size);

Hàm cấp phát vùng nhớ động kích thước size, rồi trả về địa chỉ của ngăn nhớ đầu tiên của vùng nhớ đó. Hàm trả về NULL nếu cấp phát không thành công.

* Hàm calloc(): void\*calloc(int nltems, int size\_item);

Cấp phát vùng nhớ kích thước nltems\*size\_item.

* Hàm realloc(): void\*realloc(void\* block, int new\_size);

Cấp phát lại một vùng nhớ mới kích thước new\_size cho vùng nhớ cũ trong block, có copy dữ liệu từ vùng nhớ cũ sang vùng nhớ mới.

* Hàm free(): void\* free(void\*p);

Hàm giải phóng vùng nhớ động đã được cấp phát tại địa chỉ p.

* Kiểu mảng
* Một đối tượng dữ liệu kiểu mảng đặc trưng bởi:
* Số chiểu: là một giá trị nguyên dương cố định nên được biểu diễn bằng một số nguyên dương.
* Kích thước mỗi chiều: cũng là một giá trị nguyen dương cố định nên cũng được biểu diễn bằng một hằng số nguyên dương.
* Kiểu phần tử mảng: là kiểu dữ liệu của mỗi phần tử .
* Mảng một chiều:
* Khai báo biến mảng:
* KiểuDL tên\_biến [N];
* KiểUdl tên\_biến [N] = {v1, v2,.., vn}
* KiểUdl tên\_biến [] = {v1, v2,.., vn}
* Khai báo hằng mảng:
* const kiểu\_dữ\_liệu tên\_biến [N] = {v1,..,vn};
* kiểu\_dữ\_liệu const tên\_biến [] ={v1,…,vn};
* Truy cập: tên\_mảng[i]
* Mảng 2 chiểu:
* Khai báo biến mảng:
* Kiểu\_dữ\_liệu tên\_biến [M][N];
* Kiểu\_dữ \_liệu tên\_biến [M][N] = {v1, .., vm\*n};
* Kiểu\_dữ\_liệu tên biến [][N] = {v1, vm\*n};
* Khai báo hằng mảng: tương tự như hằng mảng 1 chiều, việc khai báo hằng mảng 2 chiều sẽ phải thêm từ khóa const và khởi tạo giá trị ban đầu cho các phần tử của mảng.
* Truy cập: tên\_mảng[i][j]
* Tự định nghĩa kiểu dữ liệu mới kiểu mảng
* Typedef định\_nghĩa\_kiểu tên\_kiểu;
* Định nghĩa một kiểu mảng 1 chiều: typedef kiểu\_dữ\_liệu tên\_kiểu[N];
* Định nghĩa một kiểu mảng 2 chiều: typedef kiểu\_dữ\_liệu tên kiểu[M][N];

1. Hàm, tham chiếu, tham trị

* Hàm:
* Cấu trúc của 1 hàm:
* Phần đầu(header): gồm tên hàm, kiểu giá trị trả về(void hoặc một kiểu dl), danh sách các tham số(có thể rỗng).
* Phần thân(body): là khối lệnh chứa các lệnh cài đặt cho chức năng của hàm.
* Định nghĩa hàm: kiểu\_dữ\_liệu tên\_hàm (Kdl1 v1, kdl2 v2,..){ statements;}

Lưu ý : về tham số đầu ra. Trong C, tham số đóng vai trò đầu ra ( hoặc vừa đầu vào, vừa đầu ra, hoặc chỉ đầu ra) phải là kiểu con trỏ.

* Khai báo hàm: kiểu\_dữ\_liệu tên\_hàm (kdl1 v1, kdl2 v2,..);
* Gọi hàm: tên\_hàm (v1,v2,..);
* Con trỏ hàm:
* Khai báo: giả sử muốn khai báo con trỏ hàm pf mà sẽ trỏ đến hàm f có dạng: T f(T1 a1,.., Tm am);

Khi đó, pf sẽ được khai báo như sau: T (\*pf) (T1,..,Tm);

* Gán: pf=f; //gán hàm f cho con trỏ pf
* Gọi hàm qua con trỏ: pf(a1,..,am); hoặc (\*pf) (a1,..,am);
* Hàm callback
* Khái niệm: là hàm gọi đến hàm khác nhưng thông qua tham số là con trỏ hàm.
* Ý nghĩa: nhằm tham số hóa hàm sẽ được gọi
* VD: Xây dựng một hàm callback: sort(int a[], int N, void (\*f)) để sắp xếp dãy a gồm N phần tử, theo giải thuật sắp xếp được chọn bởi tham số f (là con trỏ hàm sẽ trỏ đến hàm cài đặt cho giải thuật sắp xếp được chọn).
* Tham chiếu:
* **Truyền tham chiếu** là truyền địa chỉ ô nhớ của biến, do đó khi thay đổi giá trị của biến bên trong phương thức thì giá trị của biến cũng bị thay đổi bên ngoài phương thức.
* Tham trị:
* **Truyền tham trị** là truyền giá trị của biến (không phải là địa chỉ ô nhớ), khi đó phương thức sẽ tự động tạo ra một địa chỉ ô nhớ mới để lưu trữ giá trị này, do đó nó chỉ được thay đổi trong phương thức hiện hành và giá trị của biến không bị thay đổi bên ngoài phương thức hiện hành.

1. **EMBEDDED C**
2. Sự giống và khác nhau giữa C thông thường và Embedded C.

* Embedded C là một biến thể của ngôn ngữ C, tuy nhiên, nó được tối ưu hóa để làm việc với phần cứng và các thiết bị nhúng.

1. Lưu ý khi lập trình C embedded.

* Một số **lưu** ý khi phát triển phần mềm nhúng. Đặc điểm đối với hệ thống nhúng: ROM và RAM hạn chế. Lập trình phụ thuộc phần cứng. Cần đáp ứng chính xác về thời gian (hàm xử lý ngắt, tác vụ…) Nhiều kiểu pointer (far/rom/ui/paged/…) Một số keywords và token đặc biệt (@, interrupt, tiny,..)

1. C memory management.

* Cấp phát động bộ nhớ
* Thư viện <stdlib.h> cung cấp một số hàm để quản lý vùng nhớ động:
* Hàm malloc(): có định dạng hàm:

Void\*malloc(int size);

Hàm cấp phát vùng nhớ động kích thước size, rồi trả về địa chỉ của ngăn nhớ đầu tiên của vùng nhớ đó. Hàm trả về NULL nếu cấp phát không thành công.

* Hàm calloc(): void\*calloc(int nltems, int size\_item);

Cấp phát vùng nhớ kích thước nltems\*size\_item.

* Hàm realloc(): void\*realloc(void\* block, int new\_size);

Cấp phát lại một vùng nhớ mới kích thước new\_size cho vùng nhớ cũ trong block, có copy dữ liệu từ vùng nhớ cũ sang vùng nhớ mới.

* Hàm free(): void\* free(void\*p);

Hàm giải phóng vùng nhớ động đã được cấp phát tại địa chỉ p.

1. Compilation process, toolchain, compiler, linker.

* Quá trình biên dịch (Compilation process) là quá trình chuyển đổi mã nguồn của một chương trình từ ngôn ngữ lập trình cấp cao thành mã máy mà máy tính có thể thực thi. Quá trình này bao gồm nhiều bước, bao gồm phân tích cú pháp, phân tích ngữ nghĩa, tạo mã trung gian (intermediate code), tối ưu hóa mã trung gian và cuối cùng là biên dịch thành mã máy.
* Quá trình biên dịch thường được thực hiện bởi một chương trình được gọi là trình biên dịch (compiler). Trình biên dịch nhận đầu vào là mã nguồn và tạo ra các tập tin thực thi hoặc thư viện được sử dụng để thực thi chương trình trên một máy tính hoặc môi trường cụ thể.
* Toolchain (chuỗi công cụ) là một tập hợp các công cụ phần mềm được sử dụng trong quá trình phát triển phần mềm để biên dịch, xây dựng, và triển khai ứng dụng. Toolchain bao gồm các công cụ như trình biên dịch (compiler), trình dịch ngược (decompiler), trình tạo mã (code generator), trình tối ưu hóa (optimizer), trình quản lý gói (package manager), và các công cụ khác cần thiết để quản lý và tự động hóa quy trình phát triển phần mềm.
* Mỗi loại phần mềm có thể có một toolchain riêng, và các công cụ trong toolchain thường được cấu hình để hoạt động cùng nhau một cách hợp nhất. Ví dụ, một toolchain cho việc phát triển phần mềm trên nền tảng nhúng (embedded systems) có thể bao gồm trình biên dịch C/C++, trình tạo mã flash, trình tối ưu hóa mã, và các công cụ khác liên quan đến việc phát triển phần mềm cho các thiết bị nhúng.
* Toolchain cũng có thể bao gồm các công cụ cho việc kiểm tra mã nguồn, quản lý phiên bản, kiểm thử và triển khai ứng dụng. Sự tích hợp tốt giữa các công cụ trong toolchain có thể giúp tăng hiệu suất và tiết kiệm thời gian trong quá trình phát triển phần mềm.
* Trình biên dịch (compiler) là một chương trình máy tính chuyển đổi mã nguồn từ một ngôn ngữ lập trình cấp cao thành mã máy có thể thực thi trên một máy tính. Quá trình này bao gồm các bước như phân tích cú pháp (parsing), phân tích ngữ nghĩa (semantic analysis), tạo mã trung gian (intermediate code generation), tối ưu hóa (optimization), và cuối cùng là tạo ra mã máy (code generation).
* Khi một người lập trình viết mã trong một ngôn ngữ lập trình như C, C++, Java, hoặc Python, mã đó cần được biên dịch thành mã máy trước khi máy tính có thể thực thi nó. Trình biên dịch chịu trách nhiệm đảm bảo rằng mã nguồn được chuyển đổi thành mã máy một cách chính xác và hiệu quả.
* Có nhiều loại trình biên dịch, bao gồm trình biên dịch mã nguồn mở như GCC (GNU Compiler Collection), trình biên dịch cung cấp bởi các nhà sản xuất phần mềm cụ thể như Microsoft Visual C++, và trình biên dịch đa nền tảng như LLVM (Low Level Virtual Machine).
* Trình biên dịch là một phần quan trọng của môi trường phát triển phần mềm và đóng vai trò quan trọng trong việc biên dịch mã nguồn thành các chương trình có thể chạy trên các hệ điều hành và nền tảng phần cứng khác nhau.
* Trình liên kết (linker) là một phần của quá trình biên dịch trong việc tạo ra các chương trình máy tính hoặc thư viện thực thi từ mã nguồn và mã máy. Trình liên kết chịu trách nhiệm tổng hợp các tập tin đối tượng (object files) hoặc các thư viện thành một tập tin thực thi có thể chạy trên máy tính. Khi một chương trình được biên dịch, mã nguồn được chia thành các tập tin đối tượng, mỗi tập tin đối tượng chứa mã máy cho một phần cụ thể của chương trình. Trình liên kết được sử dụng để kết hợp các tập tin đối tượng này thành một tập tin thực thi hoặc một thư viện.

Các chức năng cơ bản của trình liên kết bao gồm:

* Kết hợp các tập tin đối tượng: Trình liên kết sẽ kết hợp các tập tin đối tượng đã được biên dịch thành một tập tin thực thi hoặc một thư viện.
* Giải quyết các tham chiếu: Trong quá trình kết hợp, trình liên kết sẽ giải quyết các tham chiếu giữa các tập tin đối tượng. Điều này bao gồm việc đảm bảo rằng tất cả các hàm và biến được sử dụng được định nghĩa một cách đầy đủ.
* Xử lý các thư viện: Trình liên kết cũng có thể đảm nhận việc kết hợp các thư viện vào tập tin thực thi. Điều này cho phép chương trình sử dụng các chức năng từ các thư viện đã được xây dựng trước.
* Các trình liên kết thường được cung cấp kèm theo trình biên dịch hoặc là một công cụ độc lập. Ví dụ, trong môi trường phát triển phần mềm C/C++, trình liên kết thường được sử dụng sau khi mã nguồn đã được biên dịch bởi trình biên dịch như GCC (GNU Compiler Collection) hoặc Microsoft Visual C++.

1. Make, build system.

* Trong lĩnh vực phát triển phần mềm nhúng (embedded software development), "make" và "build system" đều đề cập đến các công cụ và quy trình được sử dụng để tự động hóa quá trình biên dịch và xây dựng các chương trình cho các hệ thống nhúng.
* Make: Make là một công cụ mạnh mẽ và phổ biến được sử dụng để tự động hóa quy trình biên dịch và xây dựng phần mềm. Trong môi trường nhúng, Make thường được sử dụng để xây dựng các dự án phần mềm từ mã nguồn được viết bằng ngôn ngữ C/C++ hoặc assembler. Make sử dụng tệp tin cấu hình được gọi là Makefile để xác định các quy tắc và phụ thuộc giữa các tập tin, cũng như các lệnh cần thiết để biên dịch và liên kết chúng thành các chương trình hoặc thư viện.
* Build System: Build system là một hệ thống toàn diện hơn so với Make, bao gồm cả Make cũng như các công cụ khác như CMake, Autotools, hoặc các hệ thống xây dựng tùy chỉnh. Build system không chỉ giúp biên dịch mã nguồn thành các tập tin thực thi, mà còn quản lý các tài nguyên, quản lý phiên bản, quản lý phụ thuộc, và thực hiện các nhiệm vụ khác như kiểm thử và triển khai. Trong môi trường nhúng, build system thường được tùy chỉnh để phù hợp với yêu cầu cụ thể của dự án và hệ thống mục tiêu.
* Cả Make và các hệ thống xây dựng khác đều là công cụ quan trọng trong quá trình phát triển phần mềm nhúng, giúp tăng hiệu suất và đảm bảo tính nhất quán của quá trình biên dịch và xây dựng. Đặc biệt, trong môi trường nhúng, việc tự động hóa quy trình này là rất quan trọng vì thường có các ràng buộc về tài nguyên và hiệu suất phải được xem xét cẩn thận.

1. 8bit/16bit/32bit machine.

* Khi nói về các máy nhúng (embedded systems), các thuật ngữ 8-bit, 16-bit và 32-bit thường được sử dụng để mô tả kiến trúc của vi xử lý (microcontroller hoặc microprocessor) được sử dụng trong hệ thống đó. Đây là một cách để chỉ ra kích thước của các thanh ghi (registers), kích thước dữ liệu mà vi xử lý có thể xử lý trong một lần lấy/lưu trữ (fetch/store), và kích thước của các lệnh.
* 8-bit machine: Máy 8-bit thường có thanh ghi, dữ liệu và lệnh có độ rộng 8-bit. Điều này có nghĩa là nó có thể xử lý dữ liệu và thực hiện các phép toán trên dữ liệu có độ rộng 8-bit một cách hiệu quả nhất. Máy 8-bit thường được sử dụng trong các ứng dụng đòi hỏi tài nguyên thấp và giá thành thấp, như các hệ thống nhúng đơn giản, điều khiển các thiết bị gia dụng, hoặc các ứng dụng đòi hỏi kích thước nhỏ và tiêu thụ điện năng thấp.
* 16-bit machine: Máy 16-bit có các thanh ghi, dữ liệu và lệnh có độ rộng 16-bit. Các vi xử lý 16-bit có thể xử lý dữ liệu với độ chính xác cao hơn và thường có các tính năng mạnh mẽ hơn so với các máy 8-bit. Chúng thường được sử dụng trong các ứng dụng cần xử lý dữ liệu phức tạp hơn, như các hệ thống điều khiển công nghiệp, các thiết bị y tế, hoặc các ứng dụng có yêu cầu đồ họa cơ bản.
* 32-bit machine: Máy 32-bit có các thanh ghi, dữ liệu và lệnh có độ rộng 32-bit. Các vi xử lý 32-bit có khả năng xử lý dữ liệu lớn hơn và thường có hiệu suất cao hơn so với các máy 8-bit và 16-bit. Chúng thường được sử dụng trong các ứng dụng đòi hỏi xử lý dữ liệu phức tạp, như các hệ thống điều khiển ô tô, điện thoại di động, các thiết bị thông minh, và các ứng dụng đòi hỏi hiệu suất cao.
* Khi phát triển phần mềm nhúng sử dụng ngôn ngữ lập trình như C, lập trình viên thường cần chú ý đến kiến trúc của vi xử lý mục tiêu để viết mã phù hợp với đó. Điều này bao gồm việc sử dụng các loại dữ liệu phù hợp với kích thước của máy, tối ưu hóa mã để tận dụng các tính năng cụ thể của vi xử lý, và sử dụng các tài nguyên một cách hiệu quả nhất.

1. Tổ chức firmware, kiến trúc phần mềm.

* "Tổ chức firmware" là cách mà phần mềm firmware được tổ chức và cấu trúc trên thiết bị nhúng. Firmware là phần mềm được lưu trữ trên bộ nhớ không thể xóa và thường được nhúng sâu vào phần cứng của thiết bị, giúp điều khiển và quản lý các chức năng của nó. Tổ chức firmware có thể bao gồm các thành phần sau:
* Hệ điều hành nhúng (Embedded Operating System): Trong một số trường hợp, thiết bị nhúng có thể chạy trên một hệ điều hành nhúng như FreeRTOS, Micrium uC/OS, hoặc một phiên bản nhỏ của Linux. Hệ điều hành này cung cấp các dịch vụ như quản lý tiến trình, quản lý bộ nhớ, và truy cập vào phần cứng.
* Các thành phần phần mềm cụ thể: Firmware thường được chia thành các thành phần phần mềm như trình điều khiển (drivers) phần cứng, giao diện người dùng (user interface), các phần mềm ứng dụng cụ thể, và các thư viện hỗ trợ.
* Cấu trúc thư mục và tập tin: Firmware thường có một cấu trúc thư mục và tập tin cụ thể, với các thư mục và tập tin được tổ chức một cách logic để dễ dàng quản lý và duy trì. Cấu trúc này thường phản ánh các chức năng và modules của thiết bị.
* Quy trình xây dựng và biên dịch: Firmware thường được xây dựng thông qua một quy trình tự động hóa, bao gồm việc sử dụng các công cụ như Make hoặc CMake để biên dịch mã nguồn thành mã máy và tạo các tập tin thực thi.
* Cơ chế cập nhật firmware (Firmware update mechanism): Một phần quan trọng của tổ chức firmware là cơ chế cập nhật firmware, cho phép thiết bị nhúng được cập nhật với phiên bản phần mềm mới một cách dễ dàng và an toàn.
* Tổ chức firmware có thể thay đổi tùy thuộc vào yêu cầu cụ thể của thiết bị và ứng dụng. Điều quan trọng là tổ chức phải được thiết kế để dễ dàng duy trì, mở rộng và cập nhật khi cần thiết.