**BÀI BÁO TÌM HIỂU TUẦN 1**

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

* Ngôn ngữ C là một trong những ngôn ngữ cơ bản, sơ cấp cho người mới bắt đầu làm quen với lập trình. Loại ngôn ngữ này cung cấp cho người học quyền truy cập cấp thấp vào bộ nhớ của hệ thống.
* Ưu điểm :

+ Khối xây dựng cho nhiều ngôn ngữ lập trình khác: Nhiều ngôn ngữ lập trình khác nhau như Python, C++, Java …. được xây dựng trên nền tảng C .

+ Mạnh mẽ hiệu quả : chứa nhiều kiểu dữ liệu và toán tử để cung cấp nền tảng rộng lớn.

+ Ngôn ngữ di động: C rất linh hoạt có thể giúp bạn chạy mã của mình trên bất kỳ máy nào mà không cần thực hiện thay đổi nào

+ Chúc năng tích hợp sẵn: Chỉ có 32 từ khóa trong ANSI C, có rất nhiều chức năng cài sẵn

+ Chất lượng để tự mở rộng : có thể thêm chức năng của riêng mình vào thư viện chuẩn C

+ Có cấu trúc : có thể chia thành các khối các chức năng nhỏ để dễ bảo trì kiếm tra

+ Ngôn ngữ cấp độ trung bình : có hỗ trợ cấp cao như lập trình cấp thấp

+ Cấp phát bộ nhớ động : có thể tự do cấp phát bộ nhớ tại thời điểm chạy

+ Lập tình hệ thống: có thể lập trình được thực hiện cho các phần cứng

* Nhược điểm :

+ Không hỗ trợ về khái niệm về OOP

+ Thiếu tính bảo mật: Không có các tính năng bảo mật như tự động quản lý bộ nhớ như trong các ngôn ngữ cao cấp hơn.

+ Khó khăn trong việc duy trì mã nguồn lớn: Do thiên hướng tập trung vào hiệu suất, viết và duy trì mã nguồn lớn có thể phức tạp hơn so với các ngôn ngữ khác.

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

* Biến ( Variables ): là một vị trí trong bộ nhớ của máy tính được sử dụng để lưu trữ dữ liệu.

+ Cú pháp <kiểu dữ liệu> <tên biến> ;

Int sensor ;

Double a,b,c ;

* Hằng số (constant) : giá trị không đổi trong quá trình chạy của chương trình

+ Cú pháp #define Tên\_hằng\_số giá trị

Const <kiểu dữ liệu> Tên\_hằng\_số = giá trị ;

#define PI 3.14

Const PI= 3.14 ;

* Toán tử (Operator) : là ký tự đc sử dụng thực hiện các pháp toán

= \* % / …..

* Macro : là một phần của mã nguồn được định nghĩa trước bằng 1 hằng số hoặc 1 chuỗi các lệnh

#define TÊN\_MACRO giá\_trị

#define min(a, b) (((a) < (b)) ? (a) : (b))

1. **Các cấu trúc điều kiện, vòng lặp**

* Cấu trúc điều kiện IF

If(condition)

{

//Code sẽ đc thực hiện khi condition đúng

}

* Cấu trúc điều kiện “if – else”

if(condition)

{

//Code sẽ đc thực hiện khi condition đúng

}

else

{

//Code sẽ đc thực hiện khi condition sai

}

* Vòng lặp for

for (initialization; condition; increment)

{

// Code sẽ thực thi trong mỗi vòng lặp

}

* Vòng lặp while

While (condition)

{

// code sẽ được thực hiện cho đến khi condition sai

}

* Vòng lặp do – while

Do

{

// code sẽ được thực hiện ít nhất 1 lần và đến khi condition sai

}

While(condition)

1. **Struct, Union :**

* Struct : là một kiểu cấu trúc dữ liệu gồm các biến có các loại dữ liệu khác nhau dưới một tên duy nhất.

+ Ví dụ, bạn có thể sử dụng một struct để biểu diễn thông tin về một người, bao gồm tên, tuổi và địa chỉ. Điều này giúp tổ chức dữ liệu liên quan với nhau thành một đối tượng đơn lẻ để quản lý và sử dụng trong mã của bạn.

Struct student {

Int age;

Char name[50];

}

* Union : là một loại cấu trúc dữ liệu nhưng có một khả năng hoạt động khác so với "struct". Trong một "union", tất cả các thành viên chia sẻ một vùng nhớ chung và kích thước của "union" sẽ bằng kích thước của thành viên lớn nhất.

+ Union thường được sử dụng khi cần lưu trữ dữ liệu dưới nhiều dạng khác nhau nhưng chỉ cần sử dụng một dạng tại một thời điểm. Điều này giúp tiết kiệm không gian bộ nhớ so với việc sử dụng các biến độc lập.

1. **Con trỏ, mảng**

* Con trỏ( pointer) : là một biến chứa địa chỉ của một biến khác trong bộ nhớ.

+ Con trỏ cho phép truy cập và thao tác trên dữ liệu một cách linh hoạt.

+ Bằng cách sử dụng con trỏ, có thể truy cập trực tiếp đến các vùng nhớ, điều này làm cho việc quản lý bộ nhớ và thao tác dữ liệu trở nên hiệu quả hơn.

VD: char \*p ;

P = &a;

* Mảng : là một tập hợp các phần tử cùng kiểu dữ liệu được lưu trữ liên tiếp trong bộ nhớ.

+ Mỗi phần tử trong mảng có một chỉ số (index) duy nhất để truy cập.

+ Mảng cho phép lưu trữ nhiều giá trị dữ liệu có cùng kiểu dữ liệu trong một biến duy nhất.

VD : char[50];

1. **EMBEDDED C**

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

* Giống nhau :

+ Cú pháp giống nhau : đều dùng cú pháp cơ bản của ngôn ngữ lập trình C

+ Tính di động : đều cho phép bạn tạo ra các chương trình di động có thể chạy trên nhiều nền tảng khác nhau

+ Sử dụng thư viện : đều có thư viện chuẩn và thư viện bổ sung mà bạn có thể sử dụng trong các dự án của mình.

* Khác nhau :

+ Ràng buộc phần cứng : Embedded C thường phải tuân theo các ràng buộc cụ thể của phần cứng mà chương trình sẽ chạy trên đó, như vi xử lý, vi xử lý, hoặc vi điều khiển

+ Thư viện tiêu chuẩn và phụ thuộc vào hệ thống : Embedded C thường sử dụng các thư viện và hàm tùy chỉnh được thiết kế cho các vi điều khiển cụ thể hoặc hệ thống nhúng cụ thể

+ Quản lý bộ nhớ : Trong Embedded C, quản lý bộ nhớ thường được thực hiện thủ công hơn so với C thông thường.

**2.Lưu ý khi lập trình C embedded**

- Hiểu rõ phần cứng : Đảm bảo bạn có hiểu biết sâu rộng về phần cứng mà bạn đang làm việc trên đó. Điều này bao gồm hiểu cấu trúc của vi xử lý hoặc vi điều khiển, các thiết bị ngoại vi kết nối và các giao tiếp, cũng như các yêu cầu về nguồn điện và tiêu thụ năng lượng.

- Tối ưu hóa hiệu suất : Tránh sử dụng các phép toán phức tạp hoặc thư viện không cần thiết, tối ưu hóa cấu trúc dữ liệu và thuật toán, và sử dụng các kỹ thuật biên dịch thông minh như tối ưu hóa mã, inline functions, và giảm dung lượng mã.

- Quản lý bộ nhớ : Sử dụng bộ nhớ cố định (ROM) và bộ nhớ RAM một cách hiệu quả, tránh sử dụng đệm dư thừa hoặc cấp phát động bộ nhớ nếu không cần thiết.

- Xử lý lỗi : Sử dụng các kỹ thuật xử lý lỗi như ghi log, cơ chế báo cáo lỗi và cơ chế phục hồi sau lỗi để đảm bảo tính ổn định và dễ dàng sửa lỗi.

- Kiểm tra tính tương thích : Kiểm tra các chức năng, giao tiếp và yêu cầu về tốc độ để đảm bảo rằng chương trình của bạn hoạt động đúng và ổn định.

**3. C memory management :**

- Static Allocation( Phân bổ tĩnh) : là cách cung cấp bộ nhớ cho biến và mảng tại thời điểm biên dịch.

+ Biến và mảng được phân bổ tĩnh sẽ chiếm không gian bộ nhớ cố định và không thay đổi trong suốt quá trình chạy của chương trình.

+ Thích hợp cho các biến và mảng có kích thước cố định và không thay đổi.

* Dynamic Allocation( Phân bố động) là quá trình cấp phát bộ nhớ trong thời gian chạy của chương trình.

+ Trong C, bạn có thể sử dụng các hàm như malloc(), calloc() và realloc() để cấp phát bộ nhớ động.

+ Cần phải cẩn thận khi sử dụng phân bổ động để tránh rủi ro như memory leaks (rò rỉ bộ nhớ) và fragmentation (tích tụ bộ nhớ).

* Stack và Heap :

+ Trong lập trình nhúng, thường sử dụng stack và heap để quản lý bộ nhớ.

+ Stack được sử dụng cho các biến cục bộ và các tham số hàm. Việc phân bổ và giải phóng bộ nhớ trên stack là tự động và nhanh chóng.

+ Heap được sử dụng cho phân bổ bộ nhớ động. Phân bổ và giải phóng bộ nhớ trên heap cần được thực hiện một cách cẩn thận để tránh rủi ro.

**4.** **Compilation process, toolchain, compiler, linker :**

- Compiler process: là quá trình chuyển đổi mã nguồn của một chương trình thành mã máy mà máy tính có thể thực thi. Quá trình này thường gồm các bước sau:

+ Pre\_processing ( tiền xử lý ) : Trong bước này, các chỉ thị tiền xử lý (#include, #define, #ifdef, v.v.) được xử lý và thực hiện.

+ Complation( Biên dịch ) : Trong bước này, mã nguồn C/C++ được dịch sang mã ngôn ngữ máy hoặc mã gọi là mã trung gian (intermediate code) như mã assembly.

+ Asembly : Bước này chuyển mã trung gian sang ngôn ngữ máy (ví dụ: mã máy x86, ARM, v.v.). Mã ngôn ngữ máy này là những lệnh và chỉ thị mà CPU của máy tính có thể hiểu và thực thi.

+ Linking ( liên kết ) : Trong bước này, các file mã máy hoặc mã trung gian được kết hợp với các thư viện (libraries) và các hàm ngoại vi để tạo ra một file thực thi hoàn chỉnh.

* Toolchain : là một bộ công cụ được sử dụng để biên dịch, liên kết và xây dựng chương trình từ mã nguồn. Một toolchain bao gồm compiler, assembler, linker và các công cụ khác cần thiết để phát triển phần mềm.

1. Make, build system :

* Make : một công cụ dòng lệnh (command-line tool) phổ biến được sử dụng để tự động hóa quy trình xây dựng (build process) trong phát triển phần mềm.
* Các thành phần của Make :

+ Makefile: Đây là một file văn bản chứa các quy tắc và hướng dẫn cho make về cách xây dựng chương trình từ các tệp mã nguồn và tài nguyên.

+ Targets (Mục Tiêu): Được xác định trong Makefile, mỗi mục tiêu là một nhiệm vụ hoặc một file kết quả cần được tạo ra trong quá trình xây dựng.

+ Dependencies (Phụ Thuộc): Các mục tiêu có thể phụ thuộc vào các file hoặc mục tiêu khác. Nếu một file hoặc một mục tiêu phụ thuộc thay đổi, các mục tiêu liên quan sẽ được xây dựng lại tự động.

+ Commands (Lệnh): Các hướng dẫn được cung cấp để thực hiện một nhiệm vụ cụ thể.

* Build system : là một tập hợp các công cụ và quy trình được sử dụng để tự động hóa việc xây dựng chương trình từ các mã nguồn và tài nguyên của dự án phần mềm.

Chức năng chính của build system :

+ Quản lý Dependencies (Phụ Thuộc): Theo dõi các phụ thuộc giữa các file và mục tiêu, và tự động xây dựng lại các thành phần khi có sự thay đổi trong các phụ thuộc.

+ Tạo Build Configuration: Tạo các tệp cấu hình dựa trên thông tin cấu hình của dự án và các yêu cầu xây dựng.

+ Tự Động Hóa Quy Trình Xây Dựng: Thực hiện các bước xây dựng (compile, link, pack, v.v.) tự động mà không cần sự can thiệp của người phát triển.

+ Tích Hợp Các Tiện Ích: Hỗ trợ tích hợp các công cụ bên ngoài như compiler, linker, và công cụ kiểm tra mã nguồn vào quy trình xây dựng.

+ Quản Lý Phiên Bản và Phân Phối: Hỗ trợ việc tạo các phiên bản, gói cài đặt, và quản lý quy trình phân phối chương trình.

1. 8/16/32 bits machine :

* Các máy tính 8-bit, 16-bit và 32-bit là các loại máy tính dựa trên kích thước của thanh ghi chính (main register) hoặc đơn vị xử lý trung tâm (CPU) với chiều rộng dữ liệu là 8;/16/32 bits
* Ưu và Nhược Điểm:

+ Máy tính 8-bit thường tiết kiệm năng lượng và chi phí, nhưng có hạn chế về khả năng xử lý dữ liệu.

+ Máy tính 16-bit cung cấp hiệu suất tốt hơn so với máy tính 8-bit, nhưng vẫn có hạn chế trong việc xử lý dữ liệu lớn.

+ Máy tính 32-bit cung cấp khả năng xử lý dữ liệu mạnh mẽ và hiệu suất cao hơn, phù hợp cho nhiều ứng dụng đòi hỏi về tính toán phức tạp và bộ nhớ lớn.

1. Tổ chức firmware , kiến trúc phần mềm :

* Tổ chức firmware và kiến trúc phần mềm là các khía cạnh quan trọng trong việc phát triển phần mềm nhúng (embedded software)
* Tổ chức firmware :

+ Phân Tách Chức Năng: Phân chia chương trình firmware thành các module nhỏ hơn dựa trên chức năng hoặc trách nhiệm.

Mỗi module nên chịu trách nhiệm cho một chức năng cụ thể và độc lập với các module khác.

+ Tổ Chức Lớp: Sử dụng các lớp (classes) hoặc các cấu trúc dữ liệu để tổ chức mã nguồn. Tạo các lớp hoặc cấu trúc dữ liệu phản ánh các đối tượng và tính năng của hệ thống nhúng.

+ Phân Cấp Cơ Điện Tử: Sử dụng cấu trúc phân cấp để tổ chức firmware.

+ Quản Lý Tài Nguyên: Đảm bảo quản lý tài nguyên như bộ nhớ, I/O, và nguồn điện được thực hiện một cách hiệu quả và an toàn.

* Kiến trúc phần mềm :

+ Kiến Trúc Lớp: Sử dụng mô hình kiến trúc lớp để phân chia chương trình thành các phần độc lập và tái sử dụng được

* giúp dễ dàng bảo trì và mở rộng hệ thống.

+ Kiến Trúc Sự Kiện: Sử dụng mô hình kiến trúc sự kiện (event-driven architecture) để phản ánh các sự kiện và phản ứng tương ứng của hệ thống.

* Các sự kiện và xử lý sự kiện được phân tách và tổ chức một cách rõ ràng.

+ Kiến Trúc Microservices : Sử dụng mô hình kiến trúc microservices để phân chia ứng dụng thành các dịch vụ nhỏ và độc lập.

* Mỗi dịch vụ chịu trách nhiệm cho một chức năng cụ thể và có thể được phát triển, triển khai và mở rộng độc lập.

+ Kiến Trúc Đa Tầng (Multilayer Architecture): Sử dụng mô hình kiến trúc đa tầng để phân chia ứng dụng thành các tầng (layers) với mỗi tầng có trách nhiệm cụ thể.

* Các tầng thường bao gồm tầng giao diện người dùng (UI), tầng logic ứng dụng (business logic), và tầng truy cập dữ liệu (data access).

+ Kiến Trúc Service-Oriented Architecture (SOA): Sử dụng mô hình kiến trúc hướng dịch vụ (service-oriented architecture) để phát triển ứng dụng có khả năng tái sử dụng cao.

* Ứng dụng được phân chia thành các dịch vụ độc lập với các giao tiếp tiêu chuẩn giữa chúng.