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 mệnh lệnh** được phát triển từ đầu thập niên 1970 bởi Dennis Ritchie để dùng trong hệ điều hành UNIX. Từ đó, ngôn ngữ này đã lan rộng ra nhiều hệ điều hành khác và trở thành một những ngôn ngữ phổ dụng nhất.

**Ưu điểm của C:**

* **Hiệu suất cao:** C cho phép viết mã có hiệu suất cao, phù hợp cho các ứng dụng đòi hỏi tốc độ.
* **Kiểm soát phần cứng:** C cho phép truy cập trực tiếp vào phần cứng, giúp tối ưu hóa hiệu suất và chức năng.
* **Tính di động:** Mã C có thể được biên dịch và chạy trên nhiều nền tảng.
* **Nền tảng vững chắc:** Kiến thức về C cung cấp một nền tảng vững chắc để học các ngôn ngữ lập trình khác.

**Nhược điểm của C:**

* **Quản lý bộ nhớ thủ công:** C yêu cầu quản lý bộ nhớ thủ công, điều này có thể dẫn đến lỗi như rò rỉ bộ nhớ và lỗi phân đoạn.
* **Thiếu các tính năng hiện đại:** C thiếu một số tính năng hiện đại được tìm thấy trong các ngôn ngữ lập trình khác, chẳng hạn như quản lý bộ nhớ tự động (garbage collection) và lập trình hướng đối tượng (object-oriented programming) ở mức độ cao như C++ hay Java.
* **Khó học hơn so với một số ngôn ngữ khác:** Mặc dù cú pháp đơn giản, việc hiểu các khái niệm như con trỏ và quản lý bộ nhớ có thể khó khăn đối với người mới bắt đầu.

Phạm vi sử dụng của ngôn ngữ C rất rộng rãi và đa dạng, bao gồm:

**Hệ điều hành:**

* **Lõi (kernel) hệ điều hành:** Hầu hết các hệ điều hành phổ biến như Unix, Linux, và các phiên bản Windows (phần lớn) đều được viết bằng C. Tính hiệu quả và khả năng truy cập trực tiếp vào phần cứng của C làm cho nó trở thành lựa chọn lý tưởng cho việc phát triển lõi hệ điều hành.
* **Tiện ích hệ thống:** Các tiện ích hệ thống như trình quản lý tiến trình, trình quản lý bộ nhớ, và các công cụ dòng lệnh thường được viết bằng C.

**Trình biên dịch và Thông dịch viên:**

* **Phát triển trình biên dịch:** Nhiều trình biên dịch cho các ngôn ngữ lập trình khác (như C++, Pascal, v.v.) được viết bằng C. Điều này là do C cung cấp khả năng kiểm soát chi tiết cần thiết để phân tích cú pháp, tối ưu hóa và tạo mã máy.
* **Thông dịch viên:** Một số thông dịch viên (interpreters) cho các ngôn ngữ script (như Python) cũng có thể có các thành phần được viết bằng C để cải thiện hiệu suất.

**Cơ sở dữ liệu:**

* **Hệ thống quản lý cơ sở dữ liệu (DBMS):** Các hệ thống quản lý cơ sở dữ liệu phổ biến như MySQL, PostgreSQL, và Oracle (phần lớn) đều được viết bằng C/C++. Hiệu suất là yếu tố quan trọng trong các ứng dụng cơ sở dữ liệu, và C cung cấp các công cụ cần thiết để tối ưu hóa tốc độ truy vấn và xử lý dữ liệu.

**Phần mềm nhúng:**

* **Vi điều khiển (Microcontrollers):** C là ngôn ngữ lập trình chính cho các vi điều khiển, được sử dụng trong vô số thiết bị nhúng như thiết bị gia dụng, ô tô, thiết bị y tế, và hệ thống công nghiệp. Kích thước nhỏ và hiệu quả của mã C là rất quan trọng trong môi trường tài nguyên hạn chế của các hệ thống nhúng.
* **Phần mềm điều khiển thiết bị (Device Drivers):** Các trình điều khiển thiết bị, giao tiếp giữa hệ điều hành và phần cứng, thường được viết bằng C.

**Trò chơi:**

* **Game Engine:** Mặc dù có nhiều engine game hiện đại hơn, C vẫn được sử dụng để phát triển các engine game hiệu suất cao.
* **Phát triển trò chơi:** Một số trò chơi, đặc biệt là những trò chơi đòi hỏi hiệu năng cao hoặc có các yêu cầu phần cứng cụ thể, vẫn được viết bằng C hoặc kết hợp C với các ngôn ngữ khác.

**Mạng:**

* **Giao thức mạng (Network Protocols):** Nhiều giao thức mạng (ví dụ: TCP/IP, UDP) và các ứng dụng liên quan được triển khai bằng C.
* **Phần mềm máy chủ (Server Software):** Các ứng dụng máy chủ như máy chủ web, máy chủ email, và máy chủ DNS thường được viết bằng C/C++ để đảm bảo hiệu suất và khả năng mở rộng.

**Các ứng dụng khoa học và kỹ thuật:**

* **Mô phỏng (Simulations):** C được sử dụng trong các ứng dụng mô phỏng khoa học và kỹ thuật, nơi tốc độ tính toán là rất quan trọng.
* **Xử lý tín hiệu (Signal Processing):** C được sử dụng trong các ứng dụng xử lý tín hiệu số (DSP), chẳng hạn như xử lý âm thanh, hình ảnh, và video.

**Các ứng dụng khác:**

* **Ứng dụng desktop:** Mặc dù ngày nay ít phổ biến hơn, C vẫn có thể được sử dụng để phát triển các ứng dụng desktop, đặc biệt là các ứng dụng đòi hỏi hiệu suất cao.
* **Thư viện (Libraries):** Nhiều thư viện phần mềm được viết bằng C và có thể được sử dụng trong các ngôn ngữ lập trình khác.

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

**Biến (Variables):**

* **Định nghĩa:** Biến là một vùng nhớ được đặt tên trong bộ nhớ máy tính, được sử dụng để lưu trữ dữ liệu. Giá trị của biến có thể thay đổi trong quá trình thực thi chương trình.
* **Mục đích:** Dùng để lưu trữ thông tin tạm thời hoặc lâu dài trong chương trình.
* **Ví dụ (trong C):**
* int tuoi; // Khai báo một biến tên 'tuoi' kiểu số nguyên (integer)
* float diemTB; // Khai báo một biến tên 'diemTB' kiểu số thực (floating-point)

char ten[50]; // Khai báo một biến tên 'ten' kiểu mảng ký tự (chuỗi) có kích thước 50

**Hằng (Constants):**

* **Định nghĩa:** Hằng là một giá trị không đổi trong suốt quá trình thực thi chương trình. Không giống như biến, giá trị của hằng không thể thay đổi sau khi đã được gán.
* **Mục đích:** Dùng để biểu diễn các giá trị cố định, chẳng hạn như số PI, số lượng ngày trong tuần, hoặc các hằng số cấu hình.
* **Cách khai báo hằng (trong C):**

**Sử dụng từ khóa const:**

const float PI = 3.14159;

const int SO\_NGAY\_TRONG\_TUAN = 7;

**Sử dụng #define (Macro):**

#define PI 3.14159

#define SO\_NGAY\_TRONG\_TUAN 7

**Toán tử (Operators):**

* **Định nghĩa:** Toán tử là các ký hiệu hoặc từ khóa thực hiện các phép toán trên một hoặc nhiều toán hạng (biến, hằng, hoặc biểu thức).
* **Mục đích:** Thực hiện các phép tính toán, so sánh, gán giá trị, v.v.
* **Các loại toán tử phổ biến (trong C):**
  + **Toán tử số học:** + (cộng), - (trừ), \* (nhân), / (chia), % (chia lấy dư).
  + **Toán tử gán:** = (gán), += (cộng và gán), -= (trừ và gán), \*=, /=, %=.
  + **Toán tử so sánh:** == (bằng), != (khác), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).
  + **Toán tử logic:** && (AND), || (OR), ! (NOT).
  + **Toán tử bitwise:** & (AND bit), | (OR bit), ^ (XOR bit), ~ (NOT bit), << (dịch trái), >> (dịch phải).
  + **Toán tử tăng/giảm:** ++ (tăng 1), -- (giảm 1).
  + **Toán tử điều kiện (ternary operator):** ? : (ví dụ: (a > b) ? a : b // trả về a nếu a > b, ngược lại trả về b).

**Macro (Macros):**

* + **Định nghĩa:** Macro là một đoạn mã được định nghĩa bằng chỉ thị tiền xử lý (preprocessor directive) #define. Khi trình biên dịch gặp một macro, nó sẽ thay thế macro đó bằng đoạn mã đã định nghĩa trước khi thực sự biên dịch code.
  + **Mục đích:**
  + **Định nghĩa hằng:** Như đã đề cập ở trên, macro có thể được sử dụng để định nghĩa các hằng số.
  + **Thay thế code:** Macro có thể được sử dụng để thay thế các đoạn code thường xuyên được sử dụng, giúp code ngắn gọn hơn.
  + **Biên dịch có điều kiện:** Macro có thể được sử dụng để biên dịch code một cách có điều kiện (ví dụ: chỉ biên dịch một số đoạn code nhất định khi debug).
  + **Ví dụ (trong C):**
  + #define PI 3.14159
  + #define SQUARE(x) ((x) \* (x)) // Macro tính bình phương
  + int main() {
  + float radius = 5.0;
  + float area = PI \* SQUARE(radius); // Macro SQUARE được thay thế bằng ((radius) \* (radius))
  + printf("Dien tich: %f\n", area);
  + return 0;
  + }
  + **Lưu ý quan trọng về Macro:**
  + Macro là thay thế văn bản thuần túy, không có kiểm tra kiểu dữ liệu.
  + Cần cẩn thận khi sử dụng macro với các biểu thức phức tạp, vì có thể dẫn đến các lỗi không mong muốn do thứ tự thực hiện các phép toán. Thường nên đặt các tham số macro trong ngoặc đơn để tránh các vấn đề này.
  + Trong nhiều trường hợp, inline functions (trong C++) là một lựa chọn thay thế tốt hơn macro, vì chúng cung cấp kiểm tra kiểu dữ liệu và tránh được một số vấn đề tiềm ẩn của macro.

**Kiểu dữ liệu**

* + Kiểu dữ liệu là một khái niệm quan trọng để xác định loại giá trị mà một biến có thể lưu trữ và cách thức bộ nhớ được sử dụng để lưu trữ giá trị đó. C là một ngôn ngữ kiểu tĩnh, nghĩa là kiểu dữ liệu của một biến phải được khai báo trước khi sử dụng và không thể thay đổi trong suốt vòng đời của biến đó.

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

Trong lập trình, các cấu trúc điều khiển vòng lặp (loop control structures) cho phép bạn thực thi một khối mã nhiều lần, dựa trên một điều kiện cụ thể. Chúng giúp tự động hóa các tác vụ lặp đi lặp lại, làm cho chương trình hiệu quả và dễ bảo trì hơn. Ngôn ngữ C cung cấp ba cấu trúc vòng lặp chính: for, while, và do...while.

**Vòng lặp for:**

Vòng lặp for thường được sử dụng khi bạn biết trước số lần lặp cần thiết. Nó bao gồm ba phần chính, được ngăn cách bằng dấu chấm phẩy (;):

* **Khởi tạo (Initialization):** Được thực thi một lần duy nhất trước khi vòng lặp bắt đầu. Thường được sử dụng để khởi tạo biến đếm vòng lặp.
* **Điều kiện (Condition):** Được kiểm tra trước mỗi lần lặp. Nếu điều kiện là đúng (true), khối mã bên trong vòng lặp sẽ được thực thi. Nếu điều kiện là sai (false), vòng lặp sẽ kết thúc.
* **Cập nhật (Update/Increment/Decrement):** Được thực thi sau mỗi lần lặp. Thường được sử dụng để cập nhật biến đếm vòng lặp (tăng hoặc giảm).
* **Ví dụ:** In các số từ 1 đến 10:
* #include <stdio.h>
* int main() {
* for (int i = 1; i <= 10; i++) {
* printf("%d ", i);
* }
* printf("\n");
* return 0;
* }

Vòng lặp while:

Vòng lặp while thực thi một khối mã miễn là một điều kiện nhất định là đúng. Điều kiện được kiểm tra trước khi mỗi lần lặp.

**Ví dụ:** In các số từ 1 đến 10 bằng vòng lặp while:

#include <stdio.h>

int main() {

int i = 1;

while (i <= 10) {

printf("%d ", i);

i++;

}

printf("\n");

return 0;

}

**Vòng lặp do...while:**

Vòng lặp do...while tương tự như vòng lặp while, nhưng nó đảm bảo rằng khối mã bên trong vòng lặp được thực thi *ít nhất một lần*, ngay cả khi điều kiện ban đầu là sai. Điều kiện được kiểm tra *sau* khi mỗi lần lặp.

**Ví dụ:** In các số từ 1 đến 10 bằng vòng lặp do...while:

#include <stdio.h>

int main() {

int i = 1;

do {

printf("%d ", i);

i++;

} while (i <= 10);

printf("\n");

return 0;

}

**Các câu lệnh điều khiển vòng lặp bổ sung:**

* **break:** Câu lệnh break được sử dụng để thoát khỏi vòng lặp ngay lập tức, bỏ qua mọi mã còn lại trong vòng lặp và tiếp tục thực thi mã sau vòng lặp.
* #include <stdio.h>
* int main() {
* for (int i = 1; i <= 10; i++) {
* if (i == 5) {
* break; // Thoát khỏi vòng lặp khi i bằng 5
* }
* printf("%d ", i);
* }
* printf("\n"); //In ra sau khi vòng lặp kết thúc
* return 0;
* }

// Output: 1 2 3 4

**continue:** Câu lệnh continue bỏ qua phần còn lại của lần lặp hiện tại và chuyển sang lần lặp tiếp theo.

#include <stdio.h>

int main() {

for (int i = 1; i <= 10; i++) {

if (i % 2 == 0) {

continue; // Bỏ qua các số chẵn

}

printf("%d ", i);

}

printf("\n");

return 0;

}

// Output: 1 3 5 7 9

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

Struct và Union là những cấu trúc dữ liệu tự định nghĩa (user-defined data types) mạnh mẽ trong ngôn ngữ C. Chúng cho phép bạn tạo ra các kiểu dữ liệu phức tạp bằng cách kết hợp các kiểu dữ liệu cơ bản khác nhau.

**Struct (Cấu trúc):**

* **Định nghĩa:** Một struct là một kiểu dữ liệu phức hợp cho phép bạn nhóm nhiều biến có thể có các kiểu dữ liệu khác nhau lại với nhau dưới một tên duy nhất. Các biến này được gọi là **thành viên** (members) của struct.

Công dụng: struct rất hữu ích khi bạn muốn biểu diễn một thực thể phức tạp, chẳng hạn như một sinh viên (tên, tuổi, điểm), một sản phẩm (tên, giá, mô tả), hoặc một điểm trên mặt phẳng (x, y).

**Cú pháp:**

struct <tên\_struct> {

<kiểu\_dữ\_liệu> <tên\_thành\_viên\_1>;

<kiểu\_dữ\_liệu> <tên\_thành\_viên\_2>;

...

}; // Lưu ý dấu chấm phẩy ở cuố

**Ví dụ:** Định nghĩa một struct SinhVien

#include <stdio.h>

#include <string.h> // Để sử dụng strcpy

struct SinhVien {

char ten[50];

int tuoi;

float diemTrungBinh;

};

int main() {

// Khai báo một biến kiểu SinhVien

struct SinhVien sv1;

// Gán giá trị cho các thành viên

strcpy(sv1.ten, "Nguyen Van A"); // Sử dụng strcpy để gán chuỗi

sv1.tuoi = 20;

sv1.diemTrungBinh = 8.5;

// In thông tin sinh viên

printf("Ten: %s\n", sv1.ten);

printf("Tuoi: %d\n", sv1.tuoi);

printf("Diem trung binh: %.2f\n", sv1.diemTrungBinh);

return 0;

}

**Union (Hợp nhất):**

* **Định nghĩa:** Một union là một kiểu dữ liệu phức hợp tương tự như struct, nhưng tất cả các thành viên của union chia sẻ cùng một vị trí bộ nhớ. Tại bất kỳ thời điểm nào, chỉ có một thành viên của union có thể lưu trữ một giá trị hợp lệ.

Công dụng: union thường được sử dụng khi bạn muốn lưu trữ các kiểu dữ liệu khác nhau tại cùng một vị trí bộ nhớ, nhưng bạn chỉ cần sử dụng một kiểu tại một thời điểm nhất định. Điều này có thể giúp tiết kiệm bộ nhớ.

**Cú pháp:**

union <tên\_union> {

<kiểu\_dữ\_liệu> <tên\_thành\_viên\_1>;

<kiểu\_dữ\_liệu> <tên\_thành\_viên\_2>;

...

}; // Lưu ý dấu chấm phẩy ở cuối

**Ví dụ:** Định nghĩa một union DuLieu:

#include <stdio.h>

union DuLieu {

int soNguyen;

float soThuc;

char chuCai;

};

int main() {

union DuLieu duLieu;

duLieu.soNguyen = 10;

printf("So nguyen: %d\n", duLieu.soNguyen);

duLieu.soThuc = 3.14;

printf("So thuc: %.2f\n", duLieu.soThuc);

duLieu.chuCai = 'A';

printf("Chu cai: %c\n", duLieu.chuCai);

printf("So nguyen (sau khi gan chu cai): %d\n", duLieu.soNguyen); // Giá trị có thể bị thay đổi

return 0;

}

* **Giải thích:**
  + union DuLieu: Khai báo một kiểu union có tên là DuLieu.
  + int soNguyen, float soThuc, char chuCai: Các thành viên của union.
  + union DuLieu duLieu: Khai báo một biến duLieu có kiểu DuLieu.
  + Khi bạn gán giá trị cho duLieu.soThuc, giá trị trước đó của duLieu.soNguyen sẽ bị ghi đè, vì chúng chia sẻ cùng một vị trí bộ nhớ.
* **Kích thước của union:** Kích thước của một union bằng kích thước của thành viên lớn nhất của nó.

1. **Con trỏ, mảng.**

**Con trỏ (Pointers)**

* **Định nghĩa:** Con trỏ là một biến đặc biệt lưu trữ địa chỉ bộ nhớ của một biến khác. Nói cách khác, con trỏ "trỏ" đến một vị trí trong bộ nhớ.
* **Khai báo:**

<kiểu\_dữ\_liệu> \*<tên\_con\_trỏ>;

* <kiểu\_dữ\_liệu>: Kiểu dữ liệu của biến mà con trỏ sẽ trỏ đến (ví dụ: int, float, char, struct).
* \*: Toán tử "dereference" (giải tham chiếu) dùng để khai báo một con trỏ.
* <tên\_con\_trỏ>: Tên của biến con trỏ.

&: Toán tử "address-of" (lấy địa chỉ) dùng để lấy địa chỉ bộ nhớ của một biến.

Giải tham chiếu (Dereferencing): Sử dụng toán tử \* để truy cập giá trị được lưu trữ tại địa chỉ mà con trỏ trỏ đến.

printf("Gia tri cua number: %d\n", number); // In ra 10

printf("Dia chi cua number: %p\n", &number); // In ra địa chỉ của number

printf("Gia tri cua ptr: %p\n", ptr); // In ra địa chỉ mà ptr đang trỏ đến (giống địa chỉ của number)

printf("Gia tri ma ptr tro den: %d\n", \*ptr); // In ra giá trị tại địa chỉ mà ptr trỏ đến (giống giá trị của number, tức là 10)

* **Các phép toán trên con trỏ:**
  + **Tăng/giảm con trỏ:** Bạn có thể tăng hoặc giảm con trỏ để di chuyển đến các vị trí bộ nhớ liền kề. Việc tăng/giảm con trỏ không chỉ đơn thuần là cộng/trừ 1. Nó sẽ cộng/trừ một số byte tương ứng với kích thước của kiểu dữ liệu mà con trỏ trỏ đến. Ví dụ, nếu ptr là một int \* (và int có kích thước 4 byte), thì ptr++ sẽ tăng ptr lên 4 byte.
  + **So sánh con trỏ:** Bạn có thể so sánh hai con trỏ để xem chúng có trỏ đến cùng một vị trí bộ nhớ hay không.
* **Ứng dụng của con trỏ:**
  + **Truyền tham chiếu (Pass by reference):** Cho phép hàm thay đổi giá trị của biến bên ngoài hàm.
  + **Cấp phát bộ nhớ động (Dynamic memory allocation):** Sử dụng malloc(), calloc(), realloc() và free() để quản lý bộ nhớ trong quá trình chạy chương trình.
  + **Làm việc với mảng:** Con trỏ có mối quan hệ chặt chẽ với mảng.
  + **Cấu trúc dữ liệu (Data structures):** Xây dựng các cấu trúc dữ liệu phức tạp như danh sách liên kết, cây, đồ thị.
  + **Gọi lại hàm (Function pointers):** Lưu trữ địa chỉ của hàm và gọi hàm thông qua con trỏ.

**Mảng (Arrays)**

* **Định nghĩa:** Mảng là một tập hợp các phần tử có cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ.
* **Khai báo:**

<kiểu\_dữ\_liệu> <tên\_mảng>[<số\_lượng\_phần\_tử>];

**Ví dụ:**

int numbers[5]; // Khai báo một mảng gồm 5 số nguyên

numbers[0] = 10; // Gán giá trị cho phần tử đầu tiên

numbers[1] = 20; // Gán giá trị cho phần tử thứ hai

**Khởi tạo mảng:**

int numbers[5] = {1, 2, 3, 4, 5}; // Khởi tạo mảng khi khai báo

int evenNumbers[] = {2, 4, 6, 8}; // Khởi tạo mảng mà không cần chỉ định kích thước (kích thước được suy ra từ số lượng phần tử)

**Truy cập các phần tử mảng:** Sử dụng chỉ số (index) để truy cập các phần tử mảng. Chỉ số bắt đầu từ 0.

printf("Phan tu dau tien: %d\n", numbers[0]); // In ra 10

printf("Phan tu cuoi cung: %d\n", numbers[4]); // In ra 5

**Mảng đa chiều:** C cung cấp mảng đa chiều (ví dụ: mảng 2 chiều, mảng 3 chiều).

int matrix[3][3]; // Mảng 2 chiều (ma trận 3x3)

matrix[0][0] = 1;

matrix[0][1] = 2;

**Mối quan hệ giữa con trỏ và mảng:**

* **Tên mảng là một con trỏ hằng:** Trong C, tên của một mảng (ví dụ: numbers) thực chất là một con trỏ hằng (constant pointer) trỏ đến phần tử đầu tiên của mảng. Điều này có nghĩa là bạn không thể thay đổi giá trị của tên mảng (ví dụ: numbers = something\_else; là không hợp lệ).
* **Truy cập các phần tử mảng bằng con trỏ:** Bạn có thể sử dụng con trỏ để truy cập các phần tử của mảng bằng cách sử dụng phép toán con trỏ.
* int numbers[5] = {1, 2, 3, 4, 5};
* int \*ptr = numbers; // ptr trỏ đến phần tử đầu tiên của mảng numbers
* printf("Phan tu dau tien: %d\n", \*ptr); // In ra 1 (tương đương numbers[0])

printf("Phan tu thu hai: %d\n", \*(ptr + 1)); // In ra 2 (tương đương numbers[1])

1. **Hàm, tham chiếu, tham trị**

Hàm, tham chiếu (pass by reference) và tham trị (pass by value) là những khái niệm cơ bản và vô cùng quan trọng trong lập trình C. Chúng quyết định cách dữ liệu được truyền vào và trả về từ các hàm, ảnh hưởng trực tiếp đến cách chương trình hoạt động và quản lý bộ nhớ.

**Hàm (Functions)**

* **Định nghĩa:** Hàm là một khối mã được đặt tên, thực hiện một tác vụ cụ thể. Hàm giúp chia chương trình lớn thành các phần nhỏ hơn, dễ quản lý, tái sử dụng và bảo trì.
* **Cú pháp:**
* <kiểu\_trả\_về> <tên\_hàm>(<danh\_sách\_tham\_số>) {
* // Khối mã thực thi
* return <giá\_trị\_trả\_về>; // Nếu kiểu\_trả\_về khác void

}

* + <kiểu\_trả\_về>: Kiểu dữ liệu của giá trị mà hàm trả về. Nếu hàm không trả về giá trị nào, kiểu trả về là void.
  + <tên\_hàm>: Tên của hàm.
  + <danh\_sách\_tham\_số>: Danh sách các tham số (parameters) mà hàm nhận vào. Mỗi tham số có kiểu dữ liệu và tên. Nếu hàm không nhận tham số nào, danh sách tham số là void hoặc để trống.
  + return <giá\_trị\_trả\_về>;: Lệnh return được sử dụng để trả về giá trị từ hàm. Kiểu của giá\_trị\_trả\_về phải khớp với <kiểu\_trả\_về> đã khai báo. Nếu kiểu trả về là void, bạn có thể sử dụng return; (không có giá trị trả về).
* **Ví dụ:**
* #include <stdio.h>
* // Hàm tính tổng hai số nguyên
* int tinhTong(int a, int b) {
* int tong = a + b;
* return tong;
* }
* // Hàm in ra lời chào
* void inLoiChao(char ten[]) {
* printf("Xin chao, %s!\n", ten);
* }
* int main() {
* int x = 5, y = 10;
* int ketQua = tinhTong(x, y);
* printf("Tong cua %d va %d la: %d\n", x, y, ketQua);
* inLoiChao("The Anh");
* return 0;

}

**Truyền tham trị (Pass by Value)**

* **Định nghĩa:** Khi truyền tham trị, một bản sao (copy) của giá trị của biến được truyền vào hàm. Hàm làm việc với bản sao này, không phải với biến gốc. Do đó, mọi thay đổi được thực hiện trên tham số bên trong hàm sẽ không ảnh hưởng đến biến gốc bên ngoài hàm.

**Truyền tham chiếu (Pass by Reference)**

* **Định nghĩa:** Khi truyền tham chiếu, địa chỉ bộ nhớ của biến được truyền vào hàm. Hàm có thể truy cập và thay đổi trực tiếp giá trị của biến gốc thông qua địa chỉ này. Để truyền tham chiếu trong C, bạn sử dụng con trỏ.

**Ví dụ về ứng dụng:**

* **Hoán đổi hai số:**
* #include <stdio.h>
* void hoanDoi(int \*a, int \*b) {
* int temp = \*a;
* \*a = \*b;
* \*b = temp;
* }
* int main() {
* int x = 5, y = 10;
* printf("Truoc khi hoan doi: x = %d, y = %d\n", x, y);
* hoanDoi(&x, &y);
* printf("Sau khi hoan doi: x = %d, y = %d\n", x, y);
* return 0;

}

1. Embedded C
2. Sự giống và khác nhau giữa C thông thường với Embedded C
3. Lưu ý khi lập trình C embedded
4. C memory management
5. Compilation process, toolchain, compiler, linker.
6. Make, build system.
7. 8bit/16bit/32bit machine
8. Tổ chức firmware, kiến trúc phần mềm.
9. Embedded C for 8051.

Tham khảo khóa học.

<https://www.youtube.com/watch?v=Fu-0Yc4ZS-w&list=PLhFjtzzUovr-YW6vlzkiUJRo88T4deV23&ab_channel=H%E1%BB%8CCIT-%C4%90I%E1%BB%86NT%E1%BB%AC>

1. Layout PCB.
2. Tìm hiểu cách vẽ mạch in bằng Altium trên youtube
3. Thực hiện vẽ Kit 8051 với các ngọai vi như: GPIO, LED, BUTTON, UART, khối nạp