Ghi chú C

**Thực hiện: Nguyễn Văn hào**

**Email:** [hao2205tb@gmail.com](mailto:hao2205tb@gmail.com)

(Tài liệu được biên xoạn lại từ nhiều nguồn và không dùng trong mục đích kinh doanh)

Mục lục

[1. Hello world! 4](#_Toc125643944)

[2. Token 4](#_Toc125643945)

[3. Kiểu dữ liệu 5](#_Toc125643946)

[3.1 Số nguyên: 6](#_Toc125643947)

[3.2 Số thực - dấu chấm động (Floating-Point) 7](#_Toc125643948)

[3.3 Kiểu Void 8](#_Toc125643949)

[Sizeof: Lấy kích thước của các kiểu giá trị. 9](#_Toc125643950)

[4. Biến 9](#_Toc125643951)

[4.1 Định nghĩa biến: 9](#_Toc125643952)

[Định nghĩa biến nghĩa là thông báo với trình biên dịch nơi và cách tạo lưu trữ cho biến đó. Cú pháp 1: kieu\_du\_lieu danh\_sach\_bien; 9](#_Toc125643953)

[Cú pháp 2: kieu\_du\_lieu ten\_bien = gia\_tri; 10](#_Toc125643954)

[Extern: khai báo biến ở file bất kì. 10](#_Toc125643955)

[Static: Khai báo một lần duy nhất trong hàm. 12](#_Toc125643956)

[Lvalue và Rvalue: 12](#_Toc125643957)

[Register: 13](#_Toc125643958)

[Volatile: 14](#_Toc125643959)

[Local và Global: 14](#_Toc125643960)

[Con trỏ 14](#_Toc125643961)

[Khái niệm: 14](#_Toc125643962)

[Khai báo: 14](#_Toc125643963)

[Con trỏ kiểu void 15](#_Toc125643964)

[Thay đổi giá trị của biến được con trỏ trỏ đến 16](#_Toc125643965)

[Phân biệt con trỏ hàm và hàm con trỏ 17](#_Toc125643966)

[Hàm con trỏ: 17](#_Toc125643967)

[Con trỏ hàm: 18](#_Toc125643968)

[Khai báo: 18](#_Toc125643969)

[Sử dụng: 18](#_Toc125643970)

[Passing function pointer 18](#_Toc125643971)

[Comparing function pointer 20](#_Toc125643972)

[Casting function pointer 21](#_Toc125643973)

[Con trỏ mảng 22](#_Toc125643974)

[Khái quát 22](#_Toc125643975)

[VD11: Chỉ định địa chỉ trong mảng thông qua dịch chuyển con trỏ. 22](#_Toc125643976)

[VD12: Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng (ứng dụng với hàm scanf). 22](#_Toc125643977)

[VD13 Truy suất phần tử trong mảng 23](#_Toc125643978)

[VD14: Truyền mảng vào hàm. 24](#_Toc125643979)

[Mảng con trỏ 26](#_Toc125643980)

[Phân biệt con trỏ mảng và mảng con trỏ 26](#_Toc125643981)

[Con trỏ chuỗi 27](#_Toc125643982)

[Con trỏ của con trỏ 28](#_Toc125643983)

[Con trỏ mảng 2 chiều 29](#_Toc125643984)

[Tạo thư viện 29](#_Toc125643985)

[Tạo thư viện với vscode: 29](#_Toc125643986)

[Chú ý: 29](#_Toc125643987)

[VD1: 30](#_Toc125643988)

[VD2: 31](#_Toc125643989)

[Nguyên mẫu hàm 32](#_Toc125643990)

[Khái niệm: 32](#_Toc125643991)

[Cách khai báo: 33](#_Toc125643992)

[Vị trí đặt của nguyên mẫu hàm: 33](#_Toc125643993)

[VD1: 33](#_Toc125643994)

[VD2: 34](#_Toc125643995)

[#ifndef, #ifdef, #define, #endif 35](#_Toc125643996)

[Tham khảo 35](#_Toc125643997)

# 1. Hello world!

**Chương trình c được viết dưới tệp có đuôi “.c”**

**Nội dung HelloWorld.c**

#include <stdio.h>

int main()

{

   /\* Day la chuong trinh C dau tien \*/

   printf("Hello, World! \n");

   return 0;

}

**Giải thích:**

* Thư viện stdio.h hỗ trợ các hàm giao tiếp với terminal như nhập giá trị, in ra màn hình.
* Chương trình được viết trong hàm main.
* Hàm Printf dùng in chuỗi ra terminal.

# 2. Token

Trong ngôn ngữ C bao gồm rất nhiều các token khác nhau và một token có thể là một từ khóa, một định danh, một hằng số, một chuỗi hoặc một ký tự.

Dấu chấm phảy ; trong C là một phần kết thúc lệnh. Thực tế mỗi lệnh trong C phải kết thúc bởi một dấu chấm phẩy. Nó thông báo phần kết thúc của một thuộc tính logic.

Ví dụ:

printf("Hello, World! \n");

return 0;

Comment trong C bắt đầu với /\* và kết thúc với ký tự \*/.

Ví dụ:

/\* Day la chuong trinh C dau tien \*/

Định danh (Identifier) trong C là một tên được sử dụng như một biến, hàm và một thành phần được người dùng định nghĩa.

* Một định danh có thể bắt đầu bởi các ký tự A đến Z, a đến z và dấu gạch dưới (\_) và số 0 đến 9.
* C không cho phép các dấu như @, $, và % trong tên định danh.
* C là ngôn ngữ phân biệt **chữ thường - chữ hoa**. Do đó, *VietJack* và *vietjack* là hai định danh khác nhau trong C.

Các từ khóa trong C:

|  |  |  |  |
| --- | --- | --- | --- |
| auto | else | long | switch |
| break | enum | register | typedef |
| case | extern | return | union |
| char | float | short | unsigned |
| const | for | signed | void |
| continue | goto | sizeof | volatile |
| default | if | static | while |
| do | int | struct | \_Packed |
| double |  |  |  |

# 3. Kiểu dữ liệu

Các kiểu biến trong C được phân chia như sau:

|  |  |
| --- | --- |
| STT | Kiểu và miêu tả |
| 1 | **Kiểu cơ bản**  Là các kiểu dữ liệu số học và bao gồm 2 kiểu chính: a) kiểu số nguyên và b) kiểu số thực dấu chấm động. |
| 2 | **Kiểu liệt kê**  Đây là các kiểu số học và được dùng để định nghĩa các biến mà nó có thể được gán trước một số lượng nhất định giá trị số nguyên qua suốt chương trình. |
| 3 | **Kiểu void**  Kiểu định danh *void* là kiểu đặc biệt thể hiện rằng không có giá trị nào. |
| 4 | **Kiểu phát triển từ cơ bản**  Bao gồm các kiểu : a) con trỏ, b) kiểu mảng, c) kiểu cấu trúc, d) kiểu union và e) kiểu function (hàm). |

## 3.1 Số nguyên:

Bảng giá trị:

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Cỡ lưu trữ** | **Dãy giá trị** |
| char | 1 byte | -128 tới 127 hoặc 0 tới 255 |
| unsigned char | 1 byte | 0 tới 255 |
| signed char | 1 byte | -128 tới 127 |
| int | 2 hoặc 4 bytes | -32,768 tới 32,767 hoặc -2,147,483,648 tới 2,147,483,647 |
| unsigned int | 2 hoặc 4 bytes | 0 tới 65,535 hoặc 0 tới 4,294,967,295 |
| short | 2 bytes | -32,768 tới 32,767 |
| unsigned short | 2 bytes | 0 tới 65,535 |
| long | 4 bytes | -2,147,483,648 tới 2,147,483,647 |
| unsigned long | 4 bytes | 0 tới 4,294,967,295 |

#include <stdio.h>

int main(){

    printf("kich thuoc kieu char: %d\n", sizeof(char));

    printf("kich thuoc kieu unsigned char: %d\n", sizeof(unsigned char));

    printf("kich thuoc kieu short %d\n", sizeof(short));

    printf("kich thuoc kieu int %d\n", sizeof(int));

    printf("kich thuoc kieu long %d\n", sizeof(long));

    printf("kich thuoc kieu long long %d\n", sizeof(long long));

    return 0;

}

Kết quả:

kich thuoc kieu char: 1

kich thuoc kieu unsigned char: 1

kich thuoc kieu short 2

kich thuoc kieu int 4

kich thuoc kieu long 4

kich thuoc kieu long long 8

## 3.2 Số thực - dấu chấm động (Floating-Point)

Bảng giá trị

|  |  |  |  |
| --- | --- | --- | --- |
| **Kiểu** | **Cỡ lưu trữ** | **Dãy giá trị** | **Độ chính xác** |
| float | 4 byte | 1.2E-38 tới 3.4E+38 | 6 vị trí thập phân |
| double | 8 byte | 2.3E-308 tới 1.7E+308 | 15 vị trí thập phân |
| long double | 10 byte | 3.4E-4932 tới 1.1E+4932 | 19 vị trí thập phân |

**Ví dụ:** Đọc kích thước, giá trị số thực dương nhỏ nhất và lớn nhất:

#include <stdio.h>

#include <float.h>

int main()

{

printf("Lop luu tru cho so thuc (float) la: %d \n", sizeof(float));

printf("Gia tri so thuc duong nho nhat la: %E\n", FLT\_MIN );

printf("Gia tri so thuc duong lon nhat la: %E\n", FLT\_MAX );

printf("Do chinh xac: %d\n", FLT\_DIG );

return 0;

}

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



## 3.3 Kiểu Void

Kiểu void xác định không có giá trị nào. Nó được sử dụng trong 3 trường hợp sau đây:

|  |  |
| --- | --- |
| **STT** | **Kiểu và miêu tả** |
| 1 | **Hàm trả về void**  Có rất nhiều hàm trong ngôn ngữ C mà không trả về dữ liệu nào và bạn có thể nói rằng đó là hàm void. Một hàm mà không trả về giá trị nào có kiểu là void. Ví dụ: **void exit (int status);** |
| 2 | **Hàm với tham số void**  Có những hàm trong C mà không chấp nhận bất kỳ tham số. Một hàm với không có tham số nào có thể chấp nhâu là một void. Ví dụ: **int rand(void);** |
| 3 | **Con trỏ tới void**  Một con trỏ có kiểu void \* đại diện cho địa chi của đối tượng, chứ không phải là một kiểu. Ví dụ hàm cấp phát bộ nhớ **void \*malloc (size\_t size);** trả về một con trỏ void có thể ép kiểu sang bất kỳ một đối tượng nào. |

## Sizeof: Lấy kích thước của các kiểu giá trị.

Biểu thức:

**sizeof(kieu\_du\_lieu);**

Trả về cỡ của đối tượng hoặc kiểu dưới dạng byte.

**Ví dụ:**

#include <stdio.h>

#include <limits.h>

int main()

{

printf("Kich co luu tru cho so nguyen (int) la: %d \n", sizeof(int));

return 0;

}

Biên dịch và chạy chương trình C trên sẽ cho kết quả:

Kich co luu tru cho so nguyen (int) la: 4

# 4. Biến

Khái niệm: biến là một tên được đưa ra đến bộ nhớ lưu trữ. Có kiểu xác định. VD: int, float, …

Tên của biến có thể bao gồm chữ cái, chữ số và dấu gạch dưới (\_), nhưng nó phải bắt đầu bằng ký tự chữ cái hoặc dấu gạch dưới.

## 4.1 Định nghĩa biến:

### Định nghĩa biến nghĩa là thông báo với trình biên dịch nơi và cách tạo lưu trữ cho biến đó. Cú pháp 1: kieu\_du\_lieu danh\_sach\_bien;

Ví dụ:

int i, j;

char ho, ten, c, ch;

float f, luong, diemthi;

double d;

### Cú pháp 2: kieu\_du\_lieu ten\_bien = gia\_tri;

Ví dụ:

extern int d = 3, f = 5; // khai bao bien d va f.

int d = 3, f = 5; // dinh nghia va khoi tao bien d va f.

byte z = 22; // dinh nghia va khoi tao bien z.

char x = 'hoclaptrinhc'; // bien x co gia tri la 'hoclaptrinhc'.

**Ghi chú:** Với định nghĩa không có giá trị khởi tạo, biến static có thể lưu trữ với giá trị NULL.

## Extern: khai báo biến ở file bất kì.

Extern là tên viết tắt của bên ngoài. Được sử dụng khi một tệp cụ thể cần truy cập một biến từ một tệp khác.

Thông thường biến extern được khai báo trong file.h vì khi muốn sử dụng bạn chỉ cần include file .h là có thể sử dụng biến.

Ví dụ:

lib.h

#ifndef LIB\_H

#define LIB\_H

#include <stdio.h>

extern int num;

void num\_add(int a);

void num\_show();

#endif

lib.c

#include "lib.h"

int num = 0;

void num\_add(int a){

    num += a;

}

void num\_show(){

    printf("num: %d\n", num);

}

main.c

#include "lib.h"

void main(){

    num\_show();

    num\_add(3);

    num\_show();

    num ++;

    num\_show();

}

Chạy chương trình:

gcc -o main.exe main.c lib.c

./main.exe

Kết quả:

num: 0

num: 3

num: 4

## Static: Khai báo một lần duy nhất trong hàm.

Khi 1 biến cục bộ được khai báo với từ khóa static. Biến sẽ chỉ được khởi tạo 1 lần duy nhất và tồn tại suốt thời gian chạy chương trình. Giá trị của nó không bị mất đi ngay cả khi kết thúc hàm.

Biến chỉ có thể được gọi và sử dụng thông qua hàm khai báo biến.

Các lần gọi hàm thứ 2 trở đi, biến không được khởi tạo lại nữa mà giá trị chính bằng giá trị tại lần trước hàm được gọi.

VD:

#include <stdio.h>

void func(){

    static int i = 0;

    i++;

    printf("i:%d\n", i);

}

void main(){

    func();

    func();

    func();

}

Kết quả:

i:1

i:2

i:3

## Lvalue và Rvalue:

(Theo <https://codecungnhau.com/su-khac-biet-cua-lvalue-va-rvalue-trong-c/>)

Trong C: bất cứ cái gì, miễn là bên trái của toán tử gán là lvalue và bên phải của phép gàn là rvalue.

Trong C++:

* lvalue là bất cứ thứ gì mà địa chỉ của nó có thể truy cập được. Điều đó có nghĩa là chúng ta có thể lấy được địa chỉ của lvalue bằng cách sử dụng toán tử &.
* rvalue là bất cứ thứ gì không phải là lvalue. Điều đó có nghĩa là chúng ta không thể lấy địa chỉ của rvalue và nó cũng không tồn tại vượt ra khỏi biểu thức đơn.
* VD:

int z = x + 1;

// sử dụng con trỏ để xét lvalue và rvalue.

int \* ptr0 = &(z);

int \* ptr1 = &(x + 1) // Compile Error

int \* ptr2 = &(1); // Compile Error

Trong ví dụ trên, z là lvalue vì có thể lấy được địa chỉ của z thông qua &(z). (x + 1) và 1 là rvalue do không lấy được địa chỉ.

## Register:

Dùng tăng hiệu năng trong lập trình nhúng.



Biến Register được lưu sẵn trên thanh ghi (Register) thay vì trên RAM. Do đó giảm bớt được thời gian truy suất dữ liệu từ RAM gửi tới thanh ghi.

Giải thích chi tiết tại: <https://gravieb.wordpress.com/2018/06/17/tu-khoa-register-trong-c/>

Lưu ý: Do số lượng các thanh ghi là không nhiều, cần hạn chế số lượng biến register.

## Volatile:

Trong cài đặt chương trình nhúng (hệ thống nhúng), chúng ta rất thường hay gặp biến báo cáo với từ khóa Volatile. Việc khai báo biến Volatile là rất cần thiết để tránh những lỗi sai khó phát hiện do tính năng tối ưu hóa của trình biên dịch.

Cách khai báo:

volatile int foo;//both this way...

int volatile foo;//... and this way is OK! Define a volatile integer variable

volatile uint8\_t \*pReg;//both this way...

uint8\_t volatile \*pReg;//... and this way is OK! Define a pointer to a volatile unsigned 8-bit integer

Trong thực tế, có 3 loại biến mà giá trị có thể bị thay đổi :

* Các thanh ghi ngoại vi  (ví dụ như cổng ra đa chức năng GPIO, cổng UART, cổng SPI,…) được ánh xạ bộ nhớ.

Giải thích chi tiết tại: <https://ktmt.github.io/blog/2013/05/09/y-nghia-cua-tu-khoa-volatile-trong-c/>

* Toàn cục bộ được truy xuất từ ​​các tiến trình con xử lý ngắt (ngắt dịch vụ thường trình)
* Biến toàn cục được truy xuất từ ​​nhiều tác vụ trong một ứng dụng đa luồng.

## Local và Global:

Các biến được xác định bên trong một khối hàm được gọi là **Biến** cục bộ (Local) và các biến được xác định bên ngoài khối hàm gọi là **Biến toàn cục (Global).**

**Biến cục bộ** chỉ có thể truy cập từ trong khối hàm và sẽ bị xoá khi kết thúc khối hàm.

**Biến toàn cục** có thể truy cập được trong toàn bộ chương trình.

# 5. Mảng

Tham khảo:

<https://quantrimang.com/hoc/mang-array-trong-cplusplus-156212>

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/mang-trong-c/do-dai-mang-2-chieu-trong-c/>

## Mảng một chiều

Khởi tạo:

Kieu Ten\_mang [ Kich\_co\_mang ];

int balance[5] = {15, 20, 25, 30, 35};

int balance[] = {15, 20, 25, 30, 35};

Sử dụng:

balance[4] = 35;

VD1:

#include <stdio.h>

void main(){

    int mat1[3] = {0, 1, 2};

    int mat2[] = {3, 4, 5};

    printf("mat1 %d %d %d\n", mat1[0], mat1[1], mat1[2]);

    int mat3[6];

    mat3[0] = 0;

    mat3[1] = 1;

    printf("mat3 %d %d %d\n", mat3[0], mat3[1], mat3[2]);

}

Kết quả:

mat1 0 1 2

mat3 0 1 -633334759

## Mảng 2 chiều

Khởi tạo

kieu\_du\_lieu ten\_mang [ x ][ y ];

Truy cập:

int val = a[2][3];

VD2:

#include <stdio.h>

void main(){

    int mat[3][4] = {{0, 1, 2, 3},

                      {4, 5, 6, 7},

                      {8, 9, 10, 11}};

    for (int i = 0; i <3; i +=1){

        for (int j = 0; j <4; j +=1){

            printf("%d ", mat[i][j]);

        }

        printf("\n");

    }

    mat[2][0] = 0;

    printf("mat[2][0] = %d", mat[2][0]);

}

Kết quả:

0 1 2 3

4 5 6 7

8 9 10 11

mat[2][0] = 0

# Con trỏ

## Khái niệm:

Con trỏ là biến lưu địa chỉ của biến khác.

## Khai báo:

**<**kiểu dữ liệu**>** **\*** **<**tên biến**>**

Sử dụng & để lấy địa chỉ biến.

**<**kiểu dữ liệu**>** **\*** **<**tên biến**> =** &<biến khác>

VD1:

#include <stdio.h>

void main(){

    int a = 0;

    int \*pt0;

    pt0 = &a;

    int \*pt1 = &a;

    printf("a: %d, &a: %d\n",a, &a);

    printf("\*pt0: %d, pt0: %d\n",\*pt0, pt0);

    printf("\*pt1: %d, pt1: %d\n",\*pt1, pt1);

}

Kết quả:

a: 0, &a: 1021310940

\*pt0: 0, pt0: 1021310940

\*pt1: 0, pt1: 1021310940

## Con trỏ kiểu void

Là 1 kiểu đặc biệt của con trỏ. Con trỏ kiểu void có thể trỏ đến bất kỳ đối tượng nào (với bất kỳ kiểu dữ liệu nào) có địa chỉ cụ thể trên bộ nhớ ảo.

Nên sử dụng con trỏ void vì tính linh hoạt của nó.

VD2:

#include <stdlib.h>

void main() {

    int a = 0;

    void \*pt0;

    pt0 = &a;

    void \*pt1 = &a;

    void \*pt2 = pt1;

    printf("a: %d, &a: %d\n",a, &a);

    printf("\*pt0: %d, pt0: %d\n",\*((int\*)pt0), pt0);

    printf("\*pt1: %d, pt1: %d\n",\*((int\*)pt1), pt1);

// đổi sang kiểu float vẫn được.

    printf("\*pt2: %d, pt2: %d\n",\*((float\*)pt2), pt2);

}

Kết quả:

a: 0, &a: -329253948

\*pt0: 0, pt0: -329253948

\*pt1: 0, pt1: -329253948

\*pt2: 0, pt2: -329253948

## Thay đổi giá trị của biến được con trỏ trỏ đến

Lưu ý: Không dùng lệnh ++ với con trỏ.

VD3:

#include <stdlib.h>

void main() {

    int a = 0;

    int \*pt = &a;

    a ++;

    printf("a = %d, \*pt = %d\n", a, \*pt);

    \*pt += 1;

    printf("a = %d, \*pt = %d\n", a, \*pt);

    printf("before: &a = %d, pt = %d\n", &a, pt);

    \*pt ++; // không được dùng vì sẽ làm thay đổi địa chỉ pt.

    printf("after: &a = %d, pt = %d\n", &a, pt);

    printf("a = %d, \*pt = %d\n", a, \*pt);

}

Kết quả:

a = 1, \*pt = 1

a = 2, \*pt = 2

before: &a = -2021655052, pt = -2021655052

after: &a = -2021655052, pt = -2021655048

a = 2, \*pt = -2021655048

## Phân biệt con trỏ hàm và hàm con trỏ

<https://laptrinhcanban.com/c/lap-trinh-c-co-ban/ham-trong-c/con-tro-ham-trong-c/>

Khái niệm:

* Con trỏ hàm là con trỏ được tạo ra để lưu trữ địa chỉ của một hàm trong bộ nhớ máy tính.
* Hàm con trỏ là một hàm thông thường sử dụng các con trỏ làm đối số.

## Hàm con trỏ:

VD4:

#include <stdlib.h>

void add( int \*pt){

    \*pt += 1;

}

void main() {

    int a = 0;

    add(&a);

    printf("%d\n", a);

    add(&a);

    printf("%d\n", a);

}

Kết quả:

1

2

## Con trỏ hàm:

Tham khảo: <https://www.youtube.com/watch?v=5D9okfoAlKg&t=853s>

### Khai báo:

type (\*fp) ( para\_type\_1, para\_type\_2, para\_type\_3,...);

### Sử dụng:

* Cách 1: (\*fp)(para1, para2, …);
* Cách 2: fp(para1, para2, …);

VD5:

#include <stdlib.h>

int add(int x, int y) { return x + y; }

void main() {

    int (\*func1)(int, int);

    func1 = add;

    int (\*func2)(int, int) = add;

    printf("1+2=%d\n", func1(1, 2));

    printf("1+2=%d\n", (\*func2)(1, 2));

}

Kết quả:

1+2=3

1+2=3

### Passing function pointer

Truyền hàm qua hàm.

VD6:

#include <stdio.h>

int sum(int x, int y) { return x + y; }

int sub(int x, int y) { return x - y; }

int operator(int (\*func)(int, int), int x, int y) {

     return func(x, y);

}

void main(){

    printf("1+2=%d\n", operator(sum, 1, 2));

    printf("1-2=%d\n", operator(sub, 1, 2));

}

Kết quả:

1+2=3

1-2=-1

VD7: định nghĩa kiểu cho hàm con trỏ.

#include <stdio.h>

int sum(int x, int y) { return x + y; }

int sub(int x, int y) { return x - y; }

typedef int (\*cal)(int, int);

int operator(cal func, int x, int y) {

     return func(x, y);

}

void main(){

    printf("1+2=%d\n", operator(sum, 1, 2));

    printf("1-2=%d\n", operator(sub, 1, 2));

}

Kết quả:

1+2=3

1-2=-1

Returning function pointer\

VD8

#include <stdio.h>

int sum(int x, int y) { return x + y; }

typedef int (\*cal)(int, int);

cal get\_func() {

     return sum;

}

void main(){

    cal func = get\_func();

    printf("1+2=%d\n", func(1, 2));

}

Kết quả

1+2=3

### Comparing function pointer

Kiểm tra xem con trỏ có đang trỏ tới hàm nào hay đang trỏ tới NULL.

Sử dụng toán tử == và !=.

So sánh con trỏ hàm với các toán tử <, <=, >, và >=mang lại hành vi không xác định, nếu hai con trỏ không bằng nhau.

VD9:

#include <stdio.h>

int add(int x, int y) { return x + y; }

typedef int (\*fp)(int, int);

void main(){

    fp func1 = NULL;

    fp func2 = add;

    if (func1 == NULL) { printf("func1 is NULL\n");}

    else { printf("func1 is not NULL\n");}

    if (func2 == NULL) { printf("func2 is NULL\n");}

    else { printf("func2 is not NULL\n");}

    func1 = add;

    if (func1 == func2){ printf("func1 and func2 are equal\n");}

if (func1 == add) { printf("func1 are point to add\n");}

}

Kết quả

func1 is NULL

func2 is not NULL

func1 and func2 are equal

func1 are point to add

### Casting function pointer

Ép kiểu con trỏ hàm. Ứng dụng truyền con trỏ bất kỳ.

Tham khảo: <https://www.youtube.com/watch?v=UilXm6uSUyc>

VD10

#include <stdio.h>

typedef int (\*oneArgFunPtr) (int);

typedef int (\*twoArgFunPtr) (int, int);

int add(int a, int b){ return a + b;}

void main(){

    // định nghĩa hàm con trỏ

    oneArgFunPtr funPtr = (int(\*)(int))add;

    printf("1+2=%d\n", ((int(\*)(int, int))funPtr)(1, 2));

    // sử dụng hàm con trỏ đã định nghĩa

    oneArgFunPtr funPtr2 = (oneArgFunPtr)add;

    printf("1+2=%d\n", ((twoArgFunPtr)funPtr2)(1, 2));

    // sử dụng con trỏ void

    void\* funPtr3 = add;

    printf("1+2=%d\n", ((twoArgFunPtr)funPtr3)(1, 2));

}

Kết quả:

1+2=3

1+2=3

1+2=3

## Con trỏ mảng

### Khái quát

Con trỏ mảng biểu thị địa chỉ của điểm bắt đầu mảng trong bộ nhớ.

Gán địa chỉ mảng cũng giống hàm con trỏ: không cần ký tự &.

### VD11: Chỉ định địa chỉ trong mảng thông qua dịch chuyển con trỏ.

#include <stdio.h>

void main(){

    int nums[] = {10,20,30,44,55}, \*p;

    p = nums;

    printf("array[0] address: %d\n", p); //Địa chỉ phần tử đầu tiên

    printf("array[1] address: %d\n", p + 1); //Địa chỉ phần tử thứ hai

    printf("array[3] address: %d\n", p + 3); //Địa chỉ phần tử thứ tư

}

Kết quả:

array[0] address: -1050675008

array[1] address: -1050675004

array[3] address: -1050674996

### VD12: Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng (ứng dụng với hàm scanf).

#include <stdio.h>

int main(void){

    int nums[100], \*p;

    p = nums;

    //Nhập giá trị tại địa chỉ phần tử đầu tiên

    printf("Nhap nums[0] = ");

    scanf("%d", p);

    //Nhập giá trị tại địa chỉ phần tử thứ hai

    printf("Nhap nums[1] = ");

    scanf("%d", (p + 1));

    //Nhập giá trị tại địa chỉ phần tử thứ ba

    printf("Nhap nums[2] = ");

    scanf("%d", (p + 2));

    printf("nums[0]= %d\n", nums[0]);

    printf("nums[1]= %d\n", nums[1]);

    printf("nums[2]= %d\n", nums[2]);

    return 0;

}

Kết quả:

Nhap nums[0] = 1

Nhap nums[1] = 2

Nhap nums[2] = 3

nums[0]= 1

nums[1]= 2

nums[2]= 3

### VD13 Truy suất phần tử trong mảng

Cú pháp:

\*(p + index );

| **index** | **Truy xuất bằng mảng** | **Truy xuất bằng con trỏ** |
| --- | --- | --- |
| 0 | a[0] | \*p |
| 1 | a[1] | \*(p + 1) |
| 2 | a[2] | \*(p + 2) |
| 3 | a[3] | \*(p + 3) |
| … | … | … |
| n | a[n] | \*(p + n) |

#include <stdio.h>

int main(void){

    int nums[] = {10,20,30,44,55}, \*p;

    p = nums;

    printf("%d\n", nums[2]);

    printf("%d\n", \*(p +2) );

    printf("%d\n", nums[4]);

    printf("%d\n", \*(p +4) );

    return 0;

}

Kết quả:

30

30

55

55

### VD14: Truyền mảng vào hàm.

Truyền mảng vào hàm cần truyền cả chiều dài của mảng.

Có 2 cách truyền mảng vào hàm gồm

* Truyền mảng: int array[]
* Truyền con trỏ mảng: int \*array

VD Truyền mảng vào hàm truyền thống:

/\*Tạo hàm nhập mảng 1 chiều trong C\*/

void input\_array(int array[], int length){

//array: tên mảng

//length: độ dài mảng

for (short i = 0; i < length; i++) scanf("%d", &array[i]);

}

VD Truyền mảng vào hàm sử dụng con trỏ.

/\*Tạo hàm nhập mảng bằng con trỏ trong C\*/

void input\_array(int \*array, int length){

//array: tên mảng

//length: độ dài mảng

for (short i = 0; i < length; i++) scanf("%d", (array + i));

}

VD14: Xuất nhập mảng sử dụng con trỏ mảng.

#include <stdio.h>

/\*Tạo hàm nhập mảng bằng con trỏ trong C\*/

void input\_array(int \*array, int length){

    //array: tên mảng

    //length: độ dài mảng

    for (short i = 0; i < length; i++) {

        printf("Nhap array[%d] = ", i);

        scanf("%d", (array + i));

    }

}

/\*Tạo hàm xuất mảng bằng con trỏ trong C\*/

void show\_array(int \*array, int length){

    //array: tên mảng

    //length: độ dài mảng

    for(short i = 0; i < length; i++)  printf("%d ", \*(array +i));

    printf("\n");

}

int main(void){

    /\*Nhập mảng bằng con trỏ trong c\*/

    int n;

    printf(">>Nhap so phan tu: ");

    scanf("%d", &n);

    int array[n], \*p;

    printf(">>Nhap phan tu:\n");

    input\_array(array, n);

    /\*Xuất mảng bằng con trỏ trong c\*/

    printf(">>Mang vua nhap:\n");

    show\_array(array, n);

}

Kết quả:

>>Nhap so phan tu: 3

>>Nhap phan tu:

Nhap array[0] = 12

Nhap array[1] = 23

Nhap array[2] = 34

>>Mang vua nhap:

12 23 34

## Mảng con trỏ

### Phân biệt con trỏ mảng và mảng con trỏ

* Con trỏ mảng là con trỏ được tạo ra để lưu trữ địa chỉ của một mảng trong bộ nhớ máy tính.
* Mảng con trỏ là một mảng chứa phần tử là các con trỏ.

VD15:

#include <stdio.h>

void main(){

    int array[] = {1, 2, 3};

    int \*ptr = array;

    printf("%d\n", ptr[0]);

}

Kết quả:

1

VD16: Sử dụng mảng con trỏ để lưu trữ các chuỗi.

#include <stdio.h>

void main ()

{

   char \*hotensv[] = {

                   "Tran Hung Cuong",

                   "Ho Ngoc Ha",

                   "Nguyen Son Tung",

                   "Dam Vinh Hung",

   };

   for ( int i = 0; i < 4; i++) {

      printf("Gia tri cua hotensv[%d] = %s\n", i, hotensv[i]);

   }

}

Kết quả:

Gia tri cua hotensv[0] = Tran Hung Cuong

Gia tri cua hotensv[1] = Ho Ngoc Ha

Gia tri cua hotensv[2] = Nguyen Son Tung

Gia tri cua hotensv[3] = Dam Vinh Hung

## Con trỏ chuỗi

Chuỗi có đặc điểm giống như mảng. Các phần tử trong chuỗi tương tự các phần tử trong mảng.

VD17

#include <stdio.h>

void main(void){

    char \*ptr = "Hello";

    for (int i=0; i<5; i++) {

        printf("%c ",ptr[i]);

    }

}

Kết quả:

H e l l o

## Con trỏ của con trỏ

Chính vì con trỏ cũng là một biến, nên bản thân biến con trỏ cũng được gán một địa chỉ ở đâu đó trong bộ nhớ. Có thể tạo một con trỏ khác để lưu trữ địa chỉ của biến con trỏ này. Đây được gọi là con trỏ của con trỏ trong C.



Khái báo:

type \*\*p;

VD18

#include <stdio.h>

void main(){

    int i = 0;

    int \*p1 = &i;

    int \*\*p2 = &p1;

    i++;

    printf("i=%d, &i=%d\n", i, &i);

    printf("\*p1=%d, p1=%d, &p1 = %d\n", \*p1, p1, &p1);

    printf("\*p2=%d, p2=%d, p2 = %d\n", \*\*p2, \*p2, p2);

    \*\*p2 += 1;

    printf("i=%d, &i=%d\n", i, &i);

    printf("\*p1=%d, p1=%d, &p1 = %d\n", \*p1, p1, &p1);

    printf("\*p2=%d, p2=%d, p2 = %d\n", \*\*p2, \*p2, p2);

}

Kết quả:

i=1, &i=-719324252

\*p1=1, p1=-719324252, &p1 = -719324264

\*p2=1, p2=-719324252, p2 = -719324264

i=2, &i=-719324252

\*p1=2, p1=-719324252, &p1 = -719324264

\*p2=2, p2=-719324252, p2 = -719324264

## Con trỏ mảng 2 chiều

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/mang-trong-c/con-tro-mang-2-chieu-trong-c/>

Khi không sử dụng con trỏ, các mảng 2 chiều được xử lý với kích thước và độ dài cố định, nghĩa là số hàng và số cột trong mảng sẽ luôn luôn cố định, và mảng cũng chiếm một vùng bộ nhớ trong máy tính với một số byte nhất định. Các mảng như vậy được lưu trong bộ nhớ tĩnh, và chúng ta không thể thay đổi kích thước của mảng kiểu này sau khi tạo chúng.

Con trỏ mảng được sử dụng để cấp phát bộ nhớ động cho mảng. Từ đó có thể: xóa đi một số hàng, số cột, hoặc thêm phần tử vào mảng bằng cách thêm hàng và cột.

Có hai phương pháp **khai báo mảng 2 chiều bằng con trỏ trong c**, một là xử lý nó như một **mảng 2 chiều** và hai là coi nó như một **mảng 1 chiều** để xử lý.

Sử dụng hàm **malloc** để cấp phát bộ nhớ cho biến. Sử dụng hàm **free** để giải phóng bộ nhớ sau khi đã sử dụng xong. Cả 2 đều nằm trong tệp **stdlib.h**

### a) Khai báo mảng 2 chiều trong C dưới dạng con trỏ trong mảng 2 chiều

Các bước tạo một mảng 2 chiều với y hàng và x cột:

* Tạo một mảng mẹ gồm y phần tử chứa y hàng.
* Tại mỗi phần tử tương ứng bên trên, lưu địa chỉ của mảng chứa x phần tử cột tương ứng.

VD19

#include <stdio.h>

#include <stdlib.h>

int main(void) {

    int y , x;

    printf("Nhap so hang va so cot cua mang 2 chieu: ");

    scanf("%d", &y);

    scanf("%d", &x);

    //Khai báo con trỏ của con trỏ

    int \*\*nums1;

    /\*Cấp phát bộ nhớ động cho 1 mảng mẹ gồm y phần tử,

    để chứa địa chỉ tương ứng của y hàng trong mảng 2 chiều cần tạo\*/

    /\*Gán địa chỉ mảng mẹ vào con trỏ nums1\*/

    nums1 = malloc(sizeof(int \*) \* y);

    /\*Cấp phát bộ nhớ động cho y mảng con, mỗi mảng gồm x phần tử,

    để chứa x phần tử tương ứng trong từng hàng của mảng 2 chiều cần tạo\*/

    /\*Gán địa chỉ của các mảng này vào phần tử của mảng mẹ\*/

    for(int i = 0; i < y; i++) {

        nums1[i] = malloc(sizeof(int) \* x);

    }

    /\*Truy cập và gán giá trị lần lượt vào các phần tử của y mảng con đã tạo ở trên,

    thông qua đia chỉ của chúng được gán trong con trỏ\*/

    printf("\nNhap phan tu: \n");

    for(int i = 0; i < y; i++) {

        for(int j = 0; j < x; j++) {

            scanf("%d",&nums1[i][j]);//Gán giá trị nhập từ bàn phím vào phần tử

        }

    }

    //In và kiểm tra mảng 2 chiều vừa khai báo

    printf("Mang 2 chieu vua nhap\n");

    for(int i = 0; i < y; i++) {

        for(int j = 0; j < x; j++) {

           printf("%d ", nums1[i][j]);

        }

        printf("\n");

    }

    // Giải phóng các bộ nhớ đã dùng để lưu các phần tử sau khi đã tạo mảng

    for(int i = 0; i < y; i++) {

        free(nums1[i]);

    }

    // Giải phóng bộ nhớ đã dùng để lưu địa chỉ mảng sau khi đã tạo mảng

    free(nums1);

    return 0;

}

Kết quả:

Nhap so hang va so cot cua mang 2 chieu: 2 3

Nhap phan tu:

2 3 4

5 6 7

Mang 2 chieu vua nhap

2 3 4

5 6 7

### b) Khai báo mảng 2 chiều trong C dưới dạng con trỏ trong mảng 1 chiều

Để coi mảng 2 chiều như một mảng 1 chiều, dữ liệu ở cuối hàng trước đó sẽ được sắp xếp bên cạnh dữ liệu ở đầu hàng tiếp theo và chúng được kết nối theo thứ tự. Và chúng ta cũng chỉ cần sử dụng tới 1 con trỏ mà thôi.

VD20

#include <stdio.h>

#include <stdlib.h>

int main(void) {

    int y , x;

    printf("Nhap so hang va so cot cua mang 2 chieu: ");

    scanf("%d", &y);

    scanf("%d", &x);

    int \*nums2;

    //Cấp phát bộ nhớ động cho 1 mảng gồm y×x phần tử

    //Gán địa chỉ mảng trên vào con trỏ nums2

    nums2 = malloc(sizeof(int) \* x \* y);

    for(int i = 0; i < y; i++) {

        for(int j = 0; j < x; j++) {

             scanf("%d",&nums2[i \* x + j]);//Gán giá trị nhập từ bàn phím vào phần tử

        }

    }

    //In và kiểm tra mảng 2 chiều vừa khai báo

    printf("Mang 2 chieu vua nhap\n");

     for(int i = 0; i < y; i++) {

        for(int j = 0; j < x; j++) {

               if(j < x -1) {

                    printf("%d ", nums2[i \* x + j]);

               } else {

                    printf("%d\n", nums2[i \* x + j]);

               }

        }

    }

    // Giải phóng bộ nhớ

    free(nums2);

    return 0;

}

Kết quả:

Nhap so hang va so cot cua mang 2 chieu: 2 3

1 2 3

4 5 6

Mang 2 chieu vua nhap

1 2 3

4 5 6

### c) Sử dụng con trỏ mảng 2 chiều trong hàm

VD21

#include <stdio.h>

#include <stdlib.h>

void init\_2d\_array(int \*arr, int x, int y) {

     for(int i = 0; i < y; i++) {

          for(int j = 0; j < x; j++) {

               arr[i \* x + j] = i \* x + j; // Khởi tạo giá trị trong mảng

          }

     }

}

int main(void) {

     int x = 3, y = 5;

     int \*nums;

    //Cấp phát bộ nhớ động cho 1 mảng gồm y×x phần tử

    //Gán địa chỉ mảng trên vào con trỏ nums

     nums = malloc(sizeof(int) \* x \* y);

     init\_2d\_array(nums, 5, 3);

     for(int i = 0; i < y; i++) {

          for(int j = 0; j < x; j++) {

               if(j < x - 1) {

                    printf("%d ", nums[i \* x + j]);

               } else {

                    printf("%d\n", nums[i \* x + j]);

               }

          }

     }

     // Giải phóng bộ nhớ

     free(nums);

     return 0;

}

Kết quả

0 1 2

3 4 5

6 7 8

9 10 11

12 13 14

### d) Sử dụng con trỏ mảng 2 chiều trong struct

VD22

#include <stdio.h>

#include <stdlib.h>

typedef struct str{

    int x;

    int y;

    int \*arr;

} number;

void init\_2d\_array\_struct(number \*num) {

    for(int i = 0; i < num->y; i++) {

        for(int j = 0; j < num->x; j++) {

            num->arr[i \* num->x + j] = i \* num->x + j; // Khởi tạo struct

        }

    }

}

int main(void) {

    // Khởi tạo và khai báo struct

    number num;

    num.x = 3;

    num.y = 5;

    // Cấp phát bộ nhớ động cho 1 mảng gồm y×x phần tử

    num.arr = malloc(sizeof(int) \* num.x \* num.y);

    init\_2d\_array\_struct(&num);

    for(int i = 0; i < num.y; i++) {

        for(int j = 0; j < num.x; j++) {

            if(j < num.x - 1) {

                printf("%d ", num.arr[i \* num.x + j]);

            } else {

                printf("%d\n", num.arr[i \* num.x + j]);

            }

        }

    }

    // Giải phóng bộ nhớ

    free(num.arr);

    return 0;

}

Kết quả

0 1 2

3 4 5

6 7 8

9 10 11

12 13 14

### e) Thêm hàng, cột, xoá hàng, cột.

# Tạo thư viện

## Tạo thư viện với vscode:

### Chú ý:

Trình biên dịch c\c++ trên vscode không tự động biên dịch cả file thư viện (lib.c và lib.h) do đó, cần biên dịch và chạy chương trình bằng lệnh trên terminal.

B1: Chuyển thư mục làm việc đến thư mục chứa main.c bằng cách nhấn chuột phải vào thư mục và chọn Open in Intergrated Terminal.



B2: Trên terminal gõ 2 đoạn lệnh:

gcc -o main.exe main.c lib.c

.\main.exe

Trong đó

Gcc là biên dịch cho ngôn ngữ c. g++ biên dịch cho ngôn ngữ c++ (Đoán vậy)

main.exe là output

main.c lib.c … là input

không cần gọi file header (.h) vì trong code đã include rồi.

### VD1:

lib.h

#ifndef LIB\_H

#define LIB\_H

#include <stdio.h>

void print\_hello();

#endif

lib.c

#include "lib.h"

void print\_hello(){

    printf("hello\n");

}

main.c

#include "lib.h"

void main(){

    print\_hello();

}

Chạy chương trình:

gcc -o main.exe main.c lib.c

./main.exe

Kết quả:

hello

### VD2:

lib.h

#ifndef LIB\_H

#define LIB\_H

#include <stdio.h>

extern int num;

void num\_add(int a);

void num\_show();

#endif

lib.c

#include "lib.h"

int num = 0;

void num\_add(int a){

    num += a;

}

void num\_show(){

    printf("num: %d\n", num);

}

main.c

#include "lib.h"

void main(){

    num\_show();

    num\_add(3);

    num\_show();

    num ++;

    num\_show();

}

Chạy chương trình:

gcc -o main.exe main.c lib.c

./main.exe

Kết quả:

num: 0

num: 3

num: 4

# Nguyên mẫu hàm

## Khái niệm:

**Nguyên mẫu hàm (Function Prototype)** cung cấp cho trình biên dịch (compiler) tên của hàm, kiểu dữ liệu mà hàm trả về, số lượng các tham số của hàm (gồm kiểu dữ liệu và thứ tự của các tham số đó). Nhờ đó, hàm nguyên mẫu giúp cho trình biên dịch xác nhận các lời gọi hàm mà chưa cần định nghĩa hàm đó.

## Cách khai báo:

// Trường hợp hàm không có tham số

Kiểu\_dữ\_liệu\_trả\_về　Tên\_hàm();

// Trường hợp hàm có tham số

Kiểu\_dữ\_liệu\_trả\_về　Tên\_hàm( Kiểu\_dữ\_liệu\_1　tham\_số\_1,　...,　Kiểu\_dữ\_liệu\_n tham\_số\_n );

// Hoặc khai báo ngắn gọn, không cần ghi tên tham số

Kiểu\_dữ\_liệu\_trả\_về　Tên\_hàm( Kiểu\_dữ\_liệu\_1,　...,　Kiểu\_dữ\_liệu\_n );

## Vị trí đặt của nguyên mẫu hàm:

* Trước hàm gọi nó thực hiện.
* Trong tệp HEADER (tệp .h).

## VD1:

#include <stdio.h>

void func1();

void func2(int a, int b);

void func3(int, int);

void main(){

    func1();

    func2(1, 2);

    func3(1, 2);

}

void func1(){

    printf("func1\n");

}

void func2(int a, int b){

    int add = a + b;

    printf("func2, a+b=%d\n", add);

}

void func3(int a, int b){

    int sub = a - b;

    printf("func2, a-b=%d\n", sub);

}

Kết quả:

func1

func2, a+b=3

func2, a-b=-1

## VD2:

lib.h

#ifndef LIB\_H

#define LIB\_H

#include <stdio.h>

void func();

#endif

lib.c

#include "lib.h"

void func(){

    printf("hello");

}

main.c

#include "lib.h"

void main(){

    func();

}

Chạy chương trình:

gcc -o main.exe main.c lib.c

./main.exe

Kết quả:

hello

# #ifndef, #ifdef, #define, #endif

# Tham khảo

1. Lập trình căn bản:

<https://laptrinhcanban.com/>

<https://laptrinhcanban.com/c/lap-trinh-c-co-ban/>