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

## Khái niệm

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 |  |  |  |

## Hàm scanf

scanf() trong C là một hàm cài sẵn trong tệp tiêu đề Standard Input / Output, có chức năng nhận dữ liệu nhập từ bàn phím vào chương trình C.

Cú pháp:

scanf(\*format, &object1, &object2, … )

Trong đó:

* \*format: tập hợp **chuỗi định dạng nhập xuất** của các đối tượng cần nhập vào chương trình. Định dạng nhập xuất giúp chỉ định kiểu dữ liệu của đối tượng cần nhập vào, và dấu \* có ý nghĩa là số nhiều và có bao nhiêu đối tượng được chỉ định thì cũng sẽ có bấy nhiêu định dạng tương ứng được chỉ định. Xem thêm: [Định dạng nhập xuất trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/nhap-xuat-trong-c/dinh-dang-nhap-xuat-trong-c/).
* &object: **đối tượng (dữ liệu) cần nhập vào chương trình**. Chúng ta có thể chỉ định một hoặc là nhiều đối tượng khác nhau và nhập chúng cùng lúc từ bàn phím, và trong trường hợp có nhiều đối tượng thì cần phải đặt chúng cách nhau bởi dấu phẩy. Lưu ý đừng quên viết dấu & trước các object này.
* Ký hiệu & kết hợp với tên object nhằm biểu thị địa chỉ của biến được sử dụng để lưu giá trị của object đó trong bộ nhớ máy tính.

Lưu ý là chúng ta cần **thêm ký tự &** vào đằng trước các object trong hàm scanf, trừ trường hợp object đó thuộc kiểu mảng. Lý do là bởi mảng vốn đã biểu diễn địa chỉ rồi, nên chúng ta mới không cần thêm & để biểu thị địa chỉ nữa.

Các định dạng nhập trong hàm scanf

| **Định dạng** | **Kiểu dữ liệu** | **Ý nghĩa** |
| --- | --- | --- |
| %c | char | ký tự |
| %s | char \* | chuỗi ký tự |
| %d | int, short | Số nguyên dạng thập phân |
| %f | float | Số thực |
| %lf | double | Số thực chính xác gấp đôi |

VD1:

#include <stdio.h>

void main(){

    char str1[16], str2[16];

    printf("Enter 2 string:");

    scanf("%s%s", &str1, &str2);

    printf("%s %s\n", str1, str2);

}

Kết quả:

Enter 2 string:abc "hello you"

abc "hello

VD2:

#include <stdio.h>

void main(){

    int a;

    float b;

    printf("Enter a integer number: ");

    scanf("%d",&a);

    printf("%d\n",a);

    printf("Enter a floating number: ");

    scanf("%f",&b);

    printf("%f",b);

}

Kết quả:

Enter a integer number: 12

12

Enter a floating number: 22.23

22.230000

## Hàm printf

printf() trong C là một hàm cài sẵn trong tệp tiêu đề Standard Input / Output, có chức năng hiển thị (xuất) dữ liệu ra màn hình khi chạy chương trình C.

Cú pháp:

printf (\*format, object1, object2, … )

Trong đó:

* object: đối tượng (dữ liệu) cần xuất ra màn hình. Chúng ta có thể chỉ định một hoặc là nhiều đối tượng khác nhau và xuất chúng cùng lúc ra màn hình, và trong trường hợp có nhiều đối tượng thì cần phải đặt chúng cách nhau bởi dấu phẩy.
* \*format: tập hợp chuỗi định dạng nhập xuất của các đối tượng cần xuất ra màn hình. Định dạng nhập xuất giúp chỉ định kiểu dữ liệu của đối tượng cần nhập vào, và dấu \* có ý nghĩa là *số nhiều* và có bao nhiêu đối tượng được chỉ định thì cũng sẽ có bấy nhiêu định dạng tương ứng được chỉ định. Xem thêm: [Định dạng nhập xuất trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/nhap-xuat-trong-c/dinh-dang-nhap-xuat-trong-c/).

Hàm printf() sẽ xuất ra màn hình các đối tượng cần xuất theo đúng như định dạng format đã được chỉ định. Lưu ý là chúng ta cần **chỉ định chính xác** format của đối tượng cần xuất, nếu không thì sẽ không thể xuất ra đúng kết quả mong muốn, hoặc là sẽ xảy ra lỗi trong quá trình compile chương trình C. Ví dụ như nếu xuất ra kiểu số nguyên dạng thập phân thì phải chỉ định format là %d, hoặc kiểu chuỗi thì phải là %s chẳng hạn.

Định dạng xuất:

| **Chuỗi định dạng** | **Đại diện cho kiểu ký tự** | **Ý nghĩa** |
| --- | --- | --- |
| %c | char | Xuất ra một ký tự |
| %s | char \* | Xuất ra một chuỗi ký tự |
| %d | int, short | Xuất ra một số nguyên dưới dạng thập phân |
| %u | unsigned int, unsigned short | Xuất ra một số nguyên dưới dạng thập phân không dấu |
| %x | int, short, unsigned int, unsigned short | Xuất ra một số nguyên dưới dạng thập lục phân |
| %o | int, short,unsigned int, unsigned short | Xuất ra một số nguyên dưới dạng bát phân |
| %f | float | Xuất ra một số thực |
| %e | float | Xuất ra một số thực dưới dạng số mũ |
| %g | float | Xuất ra một số thực dưới dạng phù hợp nhất |
| %ld | long | Xuất ra số nguyên chính xác kép ở dạng thập phân |
| %lu | unsigned long | Xuất ra số nguyên chính xác kép ở dạng thập phân không dấu |
| %lo | long, unsigned long | Xuất ra số nguyên chính xác kép trong hệ bát phân |
| %lx | long, unsigned long | Xuất ra số nguyên chính xác kép ở hệ thập lục phân |
| %lf | double, unsigned long | Xuất ra số thực chính xác gấp đôi |
| %a | double | Xuất ra một số thực chính xác kép thập lục phân |

VD1:

#include <stdio.h>

void main(){

    printf("xin chào thế giới\n");

    printf("Đây là ký tự đặc biệt: #");

}

Kết quả:

xin chào thế giới

Đây là ký tự đặc biệt: #

VD2:

#include <stdio.h>

void main(){

    char str[] = "Hello world!";

    printf("%s\n", str);

    int rank = 0;

    printf("rank: %d\n", rank);

    float point = 0.1;

    printf("point: %f\n", point);

}

Kết quả:

Hello world!

rank: 0

point: 0.100000

### Ký tự đặc biệt.

| **số thứ tự** | **ký tự đặc biệt** | **Ý nghĩa** |
| --- | --- | --- |
| 1 | \a | Âm cảnh báo |
| 2 | \b | backspace |
| 3 | \n | ký tự xuống dòng (LF) |
| 4 | \r | ký tự xuống dòng (CR) |
| 5 | \f | Tách trang |
| 5 | \t | ký tự TAB |
| 5 | \t | ký tự TAB |
| 6 | \v | TAB dọc |
| 7 | \¥ | Ký tự tiền yên |
| 8 | \? | Dấu chấm hỏi |
| 9 | \\ | Dấu gạch chéo ngược |
| 10 | \' | dấu nháy đơn (‘) |
| 11 | \" | dấu nháy kép (“) |
| 12 | \0 | Null |
| 13 | \000 | ASCII ký tự hệ cơ số 8 |
| 14 | \xhh | ASCII ký tự hệ cơ số 16 |

### Cách sử dụng nâng cao của printf trong C

Bằng cách chỉ định các đối số như flags, width, .precision và length, chúng ta có thể định thêm các định dạng đầu ra nâng cao như căn trái, thêm dấu, đệm số 0 v.v.. khi xuất dữ liệu trong C bằng cách sử dụng cú pháp đầy đủ của printf trong C như Kiyoshi đã trình bày ở phần trên.

%[flags][width][.precision][length]specifier

Trong đó:

* flags chứa các chỉ định có tác dụng căn chỉnh, thêm dấu , đệm số 0 v.v..
* width: độ dài tối thiểu của kết quả xuất
* .precision : độ chính xác khi xuất số thực (số chữ số đằng sau dấu phẩy số thực sau khi làm tròn)
* length: chỉ định bởi một trong các ký tự h,l hoặc L nhằm xác định số thuộc dạng short, long hay long double
* specifier là định dạng của chuỗi kết quả

Trước hết, hãy tìm hiểu về flags. Bảng các flags có thể sử dụng trong hàm printf như sau:

| **flag** | **Ý nghĩa** |
| --- | --- |
| - | Căn trái |
| + | Thêm dấu cho số |
| 0 | Đệm số 0 cho đủ độ dài kết quả xuất |
| space | Đệm dấu cách cho đủ độ dài kết quả xuất |
| # | Chỉ định các xuất dữ liệu khác |

Tiếp theo là về width. Đây là đối số chỉ định độ dài tối thiểu của chuỗi kết quả xuất ra màn hình. Nếu như độ dài thực của chuỗi kết quả **nhỏ hơn** giá trị của **width** thì các giá trị chỉ định trong flags như **số 0** hay **dấu cách** sẽ được sử dụng để đệm cho đủ độ dài **width**.

.precision là độ chính xác khi xuất ra màn hình các số thực dấu phẩy động như float hay double. Độ chính xác này nói cách khác chính là số chữ số ở phần thập phân của số thực khi xuất. Ví dụ nếu .precision bằng 2 thì số thực 1.236 sẽ được chuyển thành 1.24 chẳng hạn.

Tiếp theo là length, đối số được chỉ định bởi một trong các ký tự h, l hoặc L nhằm xác định số thuộc dạng short, long hay long double.

Cuối cùng là specifier là định dạng của chuỗi kết quả.

### Chỉ định độ chính xác cho kiểu chấm động

#include <stdio.h>

int main(void) {

double d1 = 12.34567;

double d2 = 123.4567;

//Chỉ định phần thập phân với 2 chữ số

printf("d1:[%.2f]\n", d1); //>d1:[12.35]

printf("d2:[%.2f]\n", d2); //>d2:[123.46]

return 0;

}

### Đệm dấu cách cho đủ độ dài chuỗi kết quả

#include <stdio.h>

int main(void) {

double d = 12.34;

char str[8] = "Hello!";

// Căn phải

printf("d:[%10f]\n", d);//d:[ 12.340000]

printf("str:[%10s]\n", str);//str:[ Hello!]

// Căn trái

printf("d:[%-10f]\n", d);//d:[12.340000 ]

printf("str:[%-10s]\n", str);//str:[Hello! ]

return 0;

}

### Đệm số 0 cho đủ độ dài chuỗi kết quả

#include <stdio.h>

int main(void) {

double d = 12.34;

int num = 1234;

//Đệm số 0 cho đủ độ dài chuỗi

printf("d:[%010f]\n", d);//d:[012.340000]

printf("num:[%010d]\n", num);//num:[0000001234]

return 0;

}

### Chỉ định dấu trong số khi in

Chúng ta có thể chỉ định dấu trong số khi in kết quả trong C bằng cách sử dụng tới đối số flags + như dưới đây.

#include <stdio.h>

int main(void) {

printf("[%+5d]", 32);//[ +32]

printf("[%+5d]", -32);//[ +32]

printf("[%+8.3f]", 1.414);//[ +1.414]

return 0;

}

## Hàm puts

Dùng để xuất chuỗi ký tự

Cấu trúc:

puts(str);

Điểm khác biệt giữa puts() và printf() là trong puts() không chứa nhiều đối số phức tạp như trong hàm printf mà chỉ có 1 đối số duy nhất là chuỗi ký tự (hoặc là biến chứa chuỗi ký tự) mà thôi.

Hàm puts() sẽ tự động xuất xuống dòng sau khi kết thúc chuỗi ký tự.

VD:

#include <stdio.h>

int main()

{

//Xuất trực tiếp chuỗi ký tự

puts("Hello Vietnam");//>Hello Vietnam

//Gán chuỗi vào biến và xuất giá trị trong biến

char str[20]="Hello world!!!!";

puts(str);//>Hello world!!!!

//Sử dụng printf tương tự

printf("%s", str);//>Hello world!!!!

}

## Hàm putchar()

Hàm putchar() chỉ có thể dùng để xuất **1 ký tự** ra màn hình.

Cú pháp:

puts(char);

#include <stdio.h>

int main()

{

//Xuất trực tiếp chuỗi ký tự

putchar('A');//>A

//Gán chuỗi vào biến và xuất giá trị trong biến

char a = 'a';

putchar(a);//>a

//Kết quả tượng tự khi dùng hàm printf

printf("%c", a);//>a

}

## Hàm sprintf()

Hàm sprintf() **tạo một chuỗi đầu ra từ các biến** với kiểu và định dạng khác nhau.

Cú pháp:

sprintf (str, \*format, biến-1 , biến-2 , ...)

Trong đó

* str là chuỗi ký tự được chuẩn bị để gán giá trị của các biến đã cho vào nó.
* \*format là chuỗi định dạng, là tập hợp các định dạng của biến. Có bao nhiêu biến sử dụng thì có bấy nhiêu định dạng tương ứng của chúng được viết bên trong chuỗi định dạng. Xem thêm: [Định dạng nhập xuất trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/nhap-xuat-trong-c/dinh-dang-nhap-xuat-trong-c).

VD:

#include <stdio.h>

int main(void) {

char str1[] = "Hello";

char str2[] = "World";

char chr1 = '!';

char str3[16];

sprintf(str3, "%s %s%c\n", str1, str2, chr1);

printf("%s", str3);

return 0;

}

//Hello World!

## Mã ký tự

**Đối với máy tính** thì các chữ cái, dấu hoặc chữ số như a, 8, ! mà chúng ta sử dụng trong đời sống hàng ngày **đều là ký tự không thể hiểu được**. Để máy tính có thể hiểu chúng, con người đã xây dựng những **bảng mã ký tự** mà trong đó, mỗi ký tự của ngôn ngữ con người sẽ được gán cho một số thứ tự trong bảng mã, và máy tính thay vì đọc ký tự thì sẽ đọc **thứ tự của ký tự đó trong bảng mã**. Ví dụ như chữ a thì là thứ 1, chữ o thì là thứ 3 chẳng hạn. Và chúng ta gọi các số thứ tự trong bảng mã này theo cách thân mật là **mã ký tự trong C**.

Lại nữa, giống như có rất nhiều ngôn ngữ trong thế giới loài người thì trong máy tính cũng tồn tại rất nhiều **bảng mã ký tự** khác nhau, ví dụ như *bảng mã ASCII*, *UTF-8* hay *UTF-16* chẳng hạn. Và thứ tự các ký tự của ngôn ngữ loài người trong mỗi bảng mã ký tự này thì có thể khác nhau, dẫn đến cùng một ký tự nhưng mã ký tự của nó trong các bảng mã ký tự là hoàn toàn khác nhau.

### Lấy mã ascii của ký tự trong C

Để lấy mã ascii của ký tự trong C, tuỳ vào định dạng thập phân, bát phân hay thập lục phân cần lấy mà chúng ta sử dụng hàm in printf với tham số %d (thập phân), %o (bát phân) hoặc %x (thập lục phân) như sau:

printf("%d", char);

printf("%o", char);

printf("%x", char);

Trong đó char là ký tự cần lấy mã ascii trong C.

VD1:

#include <stdio.h>

int main(void){

    printf("%d\n", 'a');

    printf("%d\n", 'b');

    printf("%d\n", 'A');

    printf("%d\n", 'B');

    printf("%x\n", 'a');

    printf("%o\n", 'a');

    return 0;

}

Kết quả:

97

98

65

66

61

141

### Chuyển từ mã ascii sang ký tự trong C

printf("%c", '\xnum');

VD2:

#include <stdio.h>

int main(void){

  printf("%c\n", '\x61');

  printf("%c\n", '\x62');

  printf("%c\n", '\x63');

  printf("%c\n", '\x41');

  printf("%c\n", '\x42');

  printf("%c\n", '\x43');

  return 0;

}

Kết quả:

a

b

c

A

B

C

# 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. |

## 3.4 Ký tự

Ký tự trong C được định nghĩa là chữ cái được biểu diễn bởi một **mã ký tự**, ví dụ như 1 chữ, 1 chữ số, hoặc là một dấu chẳng hạn.

Ký tự được biểu diễn dưới nháy đơn ‘ ’

Ví dụ: 'a' 'D' '1' '2' ',' ' '

In ký tự ra màn hình:

printf("%c", char);

Lưu ý: ký tự tiếng việt có dấu (vd: ă, ê, …) hoặc ký tự đặc biệt (#, $, …) trong c không được coi là ký tự mà chúng được coi là chuỗi và phải được đặt trong dấu nháy kép “ ”.

## 3.5 Chuỗi

### Khái niệm

Chuỗi ký tự trong C là sự kết hợp của nhiều ký tự khác nhau, ví như cụm chữ cái abc, cụm chữ số 123, hoặc cụm dấu như < . chẳng hạn.

Ký tự là một chuỗi đặc biệt.

### Biểu diễn

Chuỗi ký tự được đặt trong dấu nháy kép “ ” .

VD:

"abc"

"Love You"

"123"  
"<?> /"

**Biểu diễn chuỗi ký tự bằng tổ hợp ký tự trong C**

"abc" <==> 'a' + 'b' + 'c' + '\0'

Giống như trên, chuỗi ký tự abc được tạo ra bởi tổ hợp các ký tự a,b,c, và ở cuối cùng là ký tự đặc biệt \0. Nói chính xác thì trong C, một chuỗi ký tự được xác định từ ký tự bắt đầu nó cho tới khi xuất hiện ký tự đặc biệt \0, và ký tự đặc biệt này có tác dụng thông báo cho chương trình biết chuỗi ký tự đã kết thúc tại vị trí nó xuất hiện.

### In ký tự ra màn hình

printf("%s", string);

### Ký tự đặc biệt, ký tự Tiếng Việt

VD01.

#include <stdio.h>

int main(void){

    printf("%s\n","ê");

    printf("%s\n","ế");

    printf("%s\n","đ");

    printf("%s\n","日");

    printf("%s\n","ﾀ");

}

Kết quả:

ê

ế

đ

日

ﾀ

### Ký tự đặc biệt

| **số thứ tự** | **ký tự đặc biệt** | **Ý nghĩa** |
| --- | --- | --- |
| 1 | \a | Âm cảnh báo |
| 2 | \b | backspace |
| 3 | \n | ký tự xuống dòng (LF) |
| 4 | \r | ký tự xuống dòng (CR) |
| 5 | \f | Tách trang |
| 5 | \t | ký tự TAB |
| 5 | \t | ký tự TAB |
| 6 | \v | TAB dọc |
| 7 | \¥ | Ký tự tiền yên |
| 8 | \? | Dấu chấm hỏi |
| 9 | \\ | Dấu gạch chéo ngược |
| 10 | \' | dấu nháy đơn (‘) |
| 11 | \" | dấu nháy kép (“) |
| 12 | \0 | Null |
| 13 | \000 | ASCII ký tự hệ cơ số 8 |
| 14 | \xhh | ASCII ký tự hệ cơ số 16 |

## 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:

### Khái niệm:

Có hai trường phái định nghĩa biến trong lập trình nói chung và biến trong C nói riêng như sau:

Trường phái đầu tiên giống như ví dụ trên, coi biến trong C giống như thẻ ghi địa chỉ của dữ liệu. Các dữ liệu được lưu giữ tại các vị trí riêng biệt trong bộ nhớ với địa chỉ khác nhau, và biến trong C là thẻ dùng để ghi địa chỉ của dữ liệu đó trong bộ nhớ. Khi sử dụng dữ liệu, chúng ta sẽ truy cập vào địa chỉ được ghi trên biến của dữ liệu đó.

Trường phái thứ hai coi biến trong C giống như một cái hộp để lưu trữ dữ liệu khi xử lý chương trình. Các dữ liệu này có thể là số hoặc chuỗi mà bạn có thể ưu trữ vào biến và sử dụng nhiều lần. Kết quả của các phép xử lý như tính toán giá trị số, chỉnh sửa chuỗi ký tự sẽ tạm thời được giữ vào biến và dùng để sử dụng cho chương trình sau này.

### 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

# Câu lệnh điều kiện

## Toán tử logic

| **Toán tử** | **Tên** | **Biểu thức** | **Ý nghĩa** |
| --- | --- | --- | --- |
| && | AND | X && Y | true nếu cả X và Y đều đúng |
| || | OR | X || Y | true nếu ít nhất một trong hai vế X hoặc Y đúng |
| ! | NOT | ! X | true nếu X sai và false nếu X đúng |

## Toán tử so sánh

| **Toán tử** | **Biểu thức điều kiện** | **Ý nghĩa** |
| --- | --- | --- |
| == | x == y | x và y bằng nhau |
| != | x != y | x và y không bằng nhau |
| > | x > y | x lớn hơn y |
| < | x < y | x nhỏ hơn y |
| >= | x >= y | x bằng hoặc lớn hơn y |
| <= | x <= y | x bằng hoặc nhỏ hơn y |

## So sánh 2 ký tự

### So sánh bằng

'a' == 'a'

// true

'a' == 'b'

// false

'a' != 'a'

// false

'a' != 'b'

// true

### So sánh không bằng

'a' < 'b'

// true

'a' <= 'A'

// false

### Để kiểm tra mã ký tự ASSCII của một ký tự, chúng ta sử dụng tới hàm printf() với định dạng %d trong C như sau:

printf("Ma ascii cua a:%d\n", 'a'); //97

printf("Ma ascii cua A:%d\n", 'A'); //65

printf("Ma ascii cua b:%d\n", 'b'); //98

## So sánh 2 chuỗi

### So sánh bằng

Khi sử dụng phép so sánh bằng để **so sánh chuỗi C**, chúng ta kiểm tra giá trị của chúng có bằng nhau hay không. Lưu ý là khi so sánh chuỗi trong C, chúng ta cần phân biệt giữa chữ hoa và chữ thường. Ví dụ:

"Hello" == "Hello"

//> True

"Hello" == "hello"

//> False

"Hello" == "Hey"

//> False

"Hello" != "Hello"

//> False

"Hello" != "hello"

//> True

### So sánh không bằng

Phép so sánh sẽ bắt đầu từ ký tự đầu tiên trong hai chuỗi, và nếu chúng giống nhau, các ký tự tiếp theo được so sánh cho tới khi xuất hiện một ký tự khác nhau đầu tiên trong hai chuỗi. Khi đó, mã ký tự ASSCII của ký tự này sẽ đại diện cho cả chuỗi và được dùng để so sánh lớn nhỏ.

### Hàm strcmp()

Hàm strcmp() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **tìm hiệu mã ký tự ASSCII giữa 2 chuỗi trong C**. Thông qua hiệu này, chúng ta có thể **so sánh 2 chuỗi trong C** và tìm ra quan hệ bằng hay là lớn nhỏ giữa chúng.

Cú pháp:

strcmp(str1, str2);

| **Biểu thức** | **Giá trị trả về** | **Kết quả** |
| --- | --- | --- |
| strcmp(str1, str2) | > 0 | str1 > str2 |
| strcmp(str1, str2) | = 0 | str1 = str2 |
| strcmp(str1, str2) | < 0 | str1 < str2 |

### Hàm strncmp()

So sánh một số ký tự chỉ định giữa 2 chuỗi

Hàm strncmp() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **tìm hiệu mã ký tự ASSCII giữa 2 chuỗi trong C**. Thông qua hiệu này, chúng ta có thể **so sánh 2 chuỗi trong C** và tìm ra quan hệ bằng hay là lớn nhỏ giữa chúng.

Cú pháp: strncmp(str1, str2,n);

Trong đó str1 và str2 là 2 chuỗi cần được so sánh, và n là phạm vi số ký tự tối đa tính từ đầu hai chuỗi được dùng để so sánh.

VD:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main(void)

{

char s1[10] = "abA";

char s2[10] = "aba";

int n = 2;

printf("Ma ascii cua a:%d\n", 'a');//Ma ascii cua a:97

printf("Ma ascii cua A:%d\n", 'A');//Ma ascii cua A:65

printf("Ma ascii cua b:%d\n", 'b');//Ma ascii cua b:98

printf("Hieu ma ascii: %d\n",strncmp(s1,s2,n));//hiệu ma ascii: 0

if(strncmp(s1,s2,2)==0){

printf("%d ky tu dau cua 2 chuoi bang nhau", n);

} else if(strncmp(s1,s2,2)>0){

printf("%d ky tu dau cua %s lon hon %s", n, s1, s2);

} else

printf("%d ky tu dau cua %s nho hon %s", n, s1, s2);

return 0;

}

Kết quả:

Ma ascii cua a:97

Ma ascii cua A:65

Ma ascii cua b:98

Hieu ma ascii: 0

2 ky tu dau cua 2 chuoi bang nhau

## Câu điều kiện If

Cấu trúc:

if(condition){

câu lệnh xử lý nếu condition là True (đúng);

}

if(condition)

câu lệnh xử lý nếu condition là True (đúng);

else

câu lệnh xử lý nếu condition là False (sai);

if(condition-1){

câu lệnh xử lý nếu condition-1 là True (đúng);

}else if(condition 2){

câu lệnh xử lý nếu condition-2 là True (đúng);

}else if(condition 3){

câu lệnh xử lý nếu condition-3 là True (đúng);

…

}else{

câu lệnh xử lý nếu tất cả các condition ở trên đều False (sai);

}

**if rút gọn trong C**

Nếu trong khối lệnh của if, if else hoặc else chỉ chứa một câu lệnh duy nhất thì chúng ta có thể lược bỏ đi cặp dấu {} và sử dụng tới **if rút gọn trong C**.

VD:

#include <stdio.h>

int main(void) {

int old = 20;

if (old < 18)

printf("%s","Em chưa 18, không được vào bar");

else

printf("%s","Trên 18, đủ tuổi vào bar");

return 0;

}

//>Trên 18, đủ tuổi vào bar

VD: chúng ta cũng có thể kết hợp cả cú pháp lệnh if trong c thông thường và if rút gọn trong C trong cùng một chỗ như sau:

#include <stdio.h>

int main(void) {

int old = 17;

if (old < 18){

printf("%s","Em chưa 18, không được vào bar\n");

printf("%s","Đủ tuổi rồi tới nha");

}else

printf("%s","Trên 18, đủ tuổi vào bar");

return 0;

}

//Em chưa 18, không được vào bar

//Đủ tuổi rồi tới nha

## Switch – case:

Cú pháp:

switch(expression){

case value1：

Xử lý nếu expression === value1 ;

…

break;

case value2:

Xử lý nếu expression === value2 ;

…

break;

case value3:

Xử lý nếu expression === value3 ;

…

break;

default:

Xử lý nếu giá trị của expression không khớp với một trong các giá trị chỉ định ;

…

}

Trong đó

* expression phải là một giá trị thuộc kiểu số, một biểu thức tính toán mà kết quả là kiểu số, hoặc là một kiểu giá trị có thể chuyển đổi về dạng số, và giá trị của nó sẽ được dùng để so sánh xem có khớp với giá trị trong các case hay không.
* case là từ khoá bắt đầu các xử lý trong trường hợp giá trị của expression khớp với giá trị chỉ định đằng sau case.
* break là lệnh dùng để dừng switch sau khi hoàn thành các xử lý ở case tìm thấy. Chúng ta cũng có thể lược bỏ lệnh break.
* default là từ khoá bắt đầu các xử lý trong trường hợp giá trị của expression không khớp với bất kỳ giá trị nào được chỉ định.

Một lưu ý là khác với các ngôn ngữ lập trình khác như Java hay JavaScript có thể tự do chỉ định kiểu giá trị cho expression , thì chúng ta chỉ có thể chỉ định expression của switch case trong C phải là một giá trị thuộc kiểu số, một biểu thức tính toán trả về kết quả thuộc kiểu só, hoặc là một kiểu class có thể chuyển đổi về dạng số.

Bởi vậy, chúng ta có thể dùng kiểu số như int, double, một biểu thức tính toán với số, hoặc là kiểu giá trị có thể chuyển đổi về dạng số như enum cho expression, nhưng lại không thể dùng trực tiếp kiểu chữ như char hay string cho expression.

Thông thường chúng ta sẽ ghi lệnh break ở vị trí cuối cùng trong các xử lý sau khoá case để kết thúc switch ngay tại case tìm thấy.Tuy nhiên thì chúng ta cũng hoàn toàn có thể bỏ đi không ghi break, và khi đó đơn giản là tất cả các xử lý sẽ được chạy cho tới cuối lệnh switch case mà thôi.

## Vòng lặp For

Cú pháp:

for (biểu-thức-khởi-tạo ; biểu-thức-điều-kiện ; biểu-thức-thay-đổi ) {

Câu lệnh 1 trong khối for ;

Câu lệnh 2 trong khối for ;

...

}

**Vòng lặp for rút gọn trong C**

Lược bỏ cặp dấu ngoặc nhọn

Nếu chỉ có một câu lệnh duy nhất trong khối lệnh của for, chúng ta có thể lược bỏ cặp dấu {} và rút gọn vòng lặp for trong C như sau:

for (int i = 1; i < 3; i++)

printf("hello %d\n" , i);

Hoặc

for (int i = 1; i < 3; i++) printf("hello %d\n" , i);

Lược bỏ biểu thức khởi tạo

Nếu chúng ta đã khởi tạo biến đếm ở ngoài vòng lặp for thì biểu thức khởi tạo có thể được rút gọn. Ví dụ như sau:

int i = 0;

for (; i < 2; i++){

printf("i=%d\n" , i);

}

Lược bỏ biểu thức điều kiện

Lưu ý trong trường hợp này, để có thể thoát vòng lặp vô hạn, chúng ta cần phải ghi thêm một lệnh break bên trong xử lý của vòng lặp.

int sum = 0;

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

sum += i;

if (sum > 5){

break;

}

}

printf("sum = %d\n" , sum);

//sum = 6

Lược bỏ biểu thức thay đổi

int i = 1;

for (; i < 100;){

printf("i= %d\n",i);

i \*= 3;

}

Phạm vi sử dụng của biến đếm của vòng lặp for

Trong Biểu thức khởi tạo của câu lệnh for, chúng ta có thể khai báo biến cũng như gán giá trị cho biến đếm, nhưng phạm vi có thể sử dụng của biến được khai báo trong Biểu thức khởi tạo chỉ nằm trong khối của câu lệnh for mà thôi.

Ví dụ trong vòng lặp for sau đây, do chúng ta đã cố sử dụng giá trị của biến đếm i ở bên ngoài vòng lặp for, nên lỗi biên dịch đã xảy ra:

for (int i = 0; i < 2; i++){

printf("i=%d\n" , i);

}

printf("i= %d\n",i);

// Lỗi biên dịch trả về

Trong trường hợp bạn muốn sử dụng giá trị biến đếm ở cả bên trong lẫn bên ngoài vòng lặp, thì thay vì khai báo biến đếm trong Biểu thức khởi tạo thì hãy khai báo nó ở bên ngoài vòng lặp. Ví dụ

int i;

for (i = 0; i < 2; i++){

printf("i=%d\n" , i);

}

printf("i= %d\n",i); // 2

Vòng lặp for trong C với nhiều biến đếm

Thông thường chúng ta chỉ sử dụng một biến đếm trong một hàm for trong C, tuy nhiên bằng cách đặt các biến đếm cách nhau bởi dấu phẩy , thì chúng ta vẫn có thể sử dụng vòng lặp for với nhiều biến đếm và thay đổi đồng thời các giá trị của chúng với cú pháp như sau:

for ( BTKT1 ; BTKT2 ; BTĐK ; BTTĐ1 ; BTTĐ2 ) {

Câu lệnh 1 trong khối for ;

Câu lệnh 2 trong khối for ;

…

}

Trong đó BTKT và BTTĐ là các biểu thức khởi tạo và biểu thức thay đổi tương ứng với các biến đếm, và BTĐK là biểu thức điều kiện của lệnh for. Lưu ý là chúng ta có số BTKT và BTTĐ tương ứng với số biến đếm, nhưng chỉ có một BTĐK chung cho các biến đếm mà thôi.

Ví dụ cụ thể, chúng ta thay đổi cùng lúc giá trị của 2 biến đếm và thực hiện vòng lặp for trong C như sau:

#include <stdio.h>

int main(void){

for (int i = 1,j= 100; i < 5; i++, j--){

printf("i = %d, j = %d\n" ,i, j);

}

}

//> i = 1, j = 100

//> i = 2, j = 99

//> i = 3, j = 98

//> i = 4, j = 97

Lại nữa, vòng lặp for với nhiều biến đếm trong C sẽ thay đổi đồng thời các giá trị của biến đếm và thực thi các lệnh trong vòng lặp. Trong trường hợp bạn muốn **lần lượt thay đổi giá trị biến đếm và thực thi vòng lặp** thì hãy sử dụng for lồng trong C để thay thế.

For lồng trong C (for trong for)

for (int i = 1; i < 4; i++){

for (int j = 1; j < 4; j++){

printf("i = %d, j = %d\n" ,i, j);

}

}

Kết quả:

i = 1, j = 1

i = 1, j = 2

i = 1, j = 3

i = 2, j = 1

i = 2, j = 2

i = 2, j = 3

i = 3, j = 1

i = 3, j = 2

i = 3, j = 3

## While

Cú pháp:

while(biểu-thức-điều-kiện) {

Câu lệnh 1 trong khối while ;

Câu lệnh 2 trong khối while ;

…

}

Cú pháp rút gọn

while(biểu-thức-điều-kiện)

câu-lệnh;

Hoặc:

while(biểu-thức-điều-kiện) câu-lệnh;

## Do – while

Thực thi câu lệnh trước rồi kiểm tra điều kiện sau.

Cú pháp:

do{

Câu lệnh 1 trong khối do…while ;

Câu lệnh 2 trong khối do…while;

…

}while( biểu-thức-điều-kiện );

## Continues

Bỏ qua lượt lặp hiện tại của vòng lặp

Chúng ta sử dụng lệnh continue để **bỏ qua lượt lặp hiện tại của vòng lặp trong C** và thực hiện các lượt lặp kế tiếp.

Ví dụ, chúng ta bỏ qua lượt lặp trong chương trình in ra màn hình các phần tử trong mảng tại vòng lặp lần thứ 3 như sau:

#include <stdio.h>

int main(void){

int nums[5] = {1,2,3,4,5};

int count = 0;

int length = sizeof(nums) / sizeof(int);

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

if (i == 3){

continue;

}

printf("%d\n", nums[i]);

count += 1;

}

printf("Số phần tử đã đếm :%d\n",count);

}

//1

//2

//3

//5

//Số phần tử đã đếm :4

## Lệnh break

Thoát khỏi vòng lặp.

Chúng ta sử dụng lệnh break để **thoát khỏi vòng lặp trong C** theo điều kiện mà bạn muốn.

Ví dụ, chúng ta dừng chương trình in ra màn hình các phần tử trong mảng tại vòng lặp thứ 3 như sau:

#include <stdio.h>

int main(void){

int nums[5] = {1,2,3,4,5};

int count = 0;

int length = sizeof(nums) / sizeof(int);

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

if (i == 3){

break;

}

printf("%d\n", nums[i]);

count += 1;

}

printf("Số phần tử đã đếm :%d\n",count);

}

//1

//2

//3

//Số phần tử đã đếm :3

# 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/>