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

[Mục lục 1](#_Toc126326992)

[1. Hello world! 8](#_Toc126326993)

[2. Token 8](#_Toc126326994)

[Khái niệm 8](#_Toc126326995)

[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. 8](#_Toc126326996)

[Comment trong C bắt đầu với /\* và kết thúc với ký tự \*/. 8](#_Toc126326997)

[Định danh (Identifier) trong C là một tên được sử dụng như một biến, hàm và một thành phần được người dùng định nghĩa. 9](#_Toc126326998)

[Các từ khóa trong C: 9](#_Toc126326999)

[Hàm scanf 9](#_Toc126327000)

[Hàm printf 11](#_Toc126327001)

[Ký tự đặc biệt. 14](#_Toc126327002)

[Cách sử dụng nâng cao của printf trong C 15](#_Toc126327003)

[Chỉ định độ chính xác cho kiểu chấm động 16](#_Toc126327004)

[Đệm dấu cách cho đủ độ dài chuỗi kết quả 16](#_Toc126327005)

[Đệm số 0 cho đủ độ dài chuỗi kết quả 17](#_Toc126327006)

[Chỉ định dấu trong số khi in 17](#_Toc126327007)

[Hàm puts 18](#_Toc126327008)

[Hàm putchar() 18](#_Toc126327009)

[Hàm sprintf() 19](#_Toc126327010)

[Mã ký tự 20](#_Toc126327011)

[Lấy mã ascii của ký tự trong C 20](#_Toc126327012)

[Chuyển từ mã ascii sang ký tự trong C 21](#_Toc126327013)

[3. Kiểu dữ liệu 22](#_Toc126327014)

[3.1 Số nguyên: 23](#_Toc126327015)

[Bảng giá trị: 23](#_Toc126327016)

[3.2 Số thực - dấu chấm động (Floating-Point) 24](#_Toc126327017)

[3.3 Kiểu Void 25](#_Toc126327018)

[3.4 Ký tự 25](#_Toc126327019)

[3.5 Chuỗi 26](#_Toc126327020)

[Khái niệm 26](#_Toc126327021)

[Biểu diễn 26](#_Toc126327022)

[In ký tự ra màn hình 26](#_Toc126327023)

[Ký tự đặc biệt, ký tự Tiếng Việt 26](#_Toc126327024)

[Ký tự đặc biệt 27](#_Toc126327025)

[Hàm Strchr(): Tìm ký tự trong chuỗi 28](#_Toc126327026)

[Hàm Strstr: tìm chuỗi con trong chuỗi 29](#_Toc126327027)

[Tìm vị trí ký tự trong chuỗi 29](#_Toc126327028)

[Tìm số lần xuất hiện của một ký tự trong một chuỗi 30](#_Toc126327029)

[So sánh 2 ký tự char 31](#_Toc126327030)

[So sánh 2 chuỗi | strcmp() 31](#_Toc126327031)

[So sánh 2 chuỗi trong C với số ký tự chỉ định| Hàm strncmp() 33](#_Toc126327032)

[Hàm isspace() | Kiểm tra một ký tự có phải là khoảng trắng hay không 34](#_Toc126327033)

[Đếm khoảng trắng 35](#_Toc126327034)

[Xóa toàn bộ ký tự trắng 36](#_Toc126327035)

[Xóa khoảng trắng đầu và cuối chuỗi 36](#_Toc126327036)

[Hàm strcpy 37](#_Toc126327037)

[Độ dài chuỗi: 38](#_Toc126327038)

[Con trỏ chuỗi 39](#_Toc126327039)

[Chèn chuỗi 40](#_Toc126327040)

[Copy chuỗi 43](#_Toc126327041)

[Nối chuỗi 47](#_Toc126327042)

[Tách chuỗi | strtok() 49](#_Toc126327043)

[Tách số trong chuỗi 52](#_Toc126327044)

[Cắt chuỗi 54](#_Toc126327045)

[Xóa ký tự trong chuỗi 56](#_Toc126327046)

[Xóa ký tự cuối 57](#_Toc126327047)

[Xoá chuỗi 58](#_Toc126327048)

[Sắp xếp các ký tự của chuỗi 59](#_Toc126327049)

[Đảo ngược ký tự trong chuỗi 61](#_Toc126327050)

[Chuyển số thành chuỗi trong C | hàm sprintf() 62](#_Toc126327051)

[Chuyển int sang string trong C | hàm itoa() 63](#_Toc126327052)

[Chuyển string sang int trong C | hàm atoi() 64](#_Toc126327053)

[Chuyển string sang long | atol(str); 65](#_Toc126327054)

[Chuyển string sang float | atoi(str); 66](#_Toc126327055)

[Chuyển chữ hoa sang chữ thường trong C | Hàm tolower() 66](#_Toc126327056)

[Chuyển chữ thường sang chữ hoa trong C | Hàm toupper() 67](#_Toc126327057)

[Tự tạo hàm chuyển chữ hoa chữ thường trong C 68](#_Toc126327058)

[Kiểm tra chữ hoa chữ thường 69](#_Toc126327059)

[Nhập chuỗi có khoảng trắng 70](#_Toc126327060)

[Hàm random chuỗi 71](#_Toc126327061)

[3.6 Kiểm tra chắn lẻ 72](#_Toc126327062)

[3.7 Kiểm tra số nguyên 73](#_Toc126327063)

[3.8 Kiểm tra số nguyên tố 75](#_Toc126327064)

[3.9 Kiểm tra số âm, số dương trong c 78](#_Toc126327065)

[3.10 Kiểm tra số chính phương 78](#_Toc126327066)

[3.11 Kiểm tra số hoàn hảo 82](#_Toc126327067)

[3.12 Kiểm tra số đối xứng 83](#_Toc126327068)

[3.13 Tìm số đảo ngược 84](#_Toc126327069)

[Sizeof: Lấy kích thước của các kiểu giá trị. 85](#_Toc126327070)

[4. Biến 85](#_Toc126327071)

[4.1 Định nghĩa biến: 85](#_Toc126327072)

[Khái niệm: 85](#_Toc126327073)

[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; 86](#_Toc126327074)

[Cú pháp 2: kieu\_du\_lieu ten\_bien = gia\_tri; 86](#_Toc126327075)

[Extern: khai báo biến ở file bất kì. 86](#_Toc126327076)

[Static: Khai báo một lần duy nhất trong hàm. 88](#_Toc126327077)

[Lvalue và Rvalue: 89](#_Toc126327078)

[Register: 89](#_Toc126327079)

[Volatile: 90](#_Toc126327080)

[Local và Global: 91](#_Toc126327081)

[5. Mảng 91](#_Toc126327082)

[Mảng một chiều 91](#_Toc126327083)

[Mảng 2 chiều 92](#_Toc126327084)

[Lấy kích thước mảng trong C | toán tử sizeof() 93](#_Toc126327085)

[Độ dài mảng 94](#_Toc126327086)

[Con trỏ mảng 95](#_Toc126327087)

[Khai báo con trỏ 95](#_Toc126327088)

[Chỉ định địa chỉ trong mảng thông qua dịch chuyển con trỏ 95](#_Toc126327089)

[Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng 96](#_Toc126327090)

[Truy xuất giá trị phần tử trong mảng bằng con trỏ 97](#_Toc126327091)

[Sao chép mảng 98](#_Toc126327092)

[Cách 1; Dùng vòng lặp for để chép từng phần tử mảng A sang mảng B . 98](#_Toc126327093)

[Cách 2: Sử dụng hàm memcpy 100](#_Toc126327094)

[Thêm phần tử vào cuối mảng 100](#_Toc126327095)

[Chèn phần tử vào mảng trong C vào vị trí bất kỳ 101](#_Toc126327096)

[Ghép 2 mảng 103](#_Toc126327097)

[Xóa 1 phần tử trong mảng 104](#_Toc126327098)

[Xoá nhiều phần tử 107](#_Toc126327099)

[Xóa toàn bộ mảng 109](#_Toc126327100)

[Tách mảng âm dương 110](#_Toc126327101)

[Tách mảng chẵn lẻ 111](#_Toc126327102)

[Câu lệnh điều kiện 112](#_Toc126327103)

[Toán tử logic 112](#_Toc126327104)

[Toán tử so sánh 113](#_Toc126327105)

[So sánh 2 ký tự 113](#_Toc126327106)

[So sánh bằng 113](#_Toc126327107)

[So sánh không bằng 114](#_Toc126327108)

[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: 114](#_Toc126327109)

[So sánh 2 chuỗi 114](#_Toc126327110)

[So sánh bằng 114](#_Toc126327111)

[So sánh không bằng 115](#_Toc126327112)

[Hàm strcmp() 115](#_Toc126327113)

[Hàm strncmp() 115](#_Toc126327114)

[Câu điều kiện If 117](#_Toc126327115)

[Switch – case: 118](#_Toc126327116)

[Vòng lặp For 120](#_Toc126327117)

[While 124](#_Toc126327118)

[Do – while 124](#_Toc126327119)

[Continues 125](#_Toc126327120)

[Lệnh break 125](#_Toc126327121)

[Con trỏ 126](#_Toc126327122)

[Khái niệm: 126](#_Toc126327123)

[Khai báo: 126](#_Toc126327124)

[Con trỏ kiểu void 127](#_Toc126327125)

[Thay đổi giá trị của biến được con trỏ trỏ đến 128](#_Toc126327126)

[Phân biệt con trỏ hàm và hàm con trỏ 129](#_Toc126327127)

[Hàm con trỏ: 129](#_Toc126327128)

[Con trỏ hàm: 130](#_Toc126327129)

[Khai báo: 130](#_Toc126327130)

[Sử dụng: 130](#_Toc126327131)

[Passing function pointer 130](#_Toc126327132)

[Comparing function pointer 132](#_Toc126327133)

[Casting function pointer 133](#_Toc126327134)

[Con trỏ mảng 134](#_Toc126327135)

[Khái quát 134](#_Toc126327136)

[VD11: Chỉ định địa chỉ trong mảng thông qua dịch chuyển con trỏ. 134](#_Toc126327137)

[VD12: Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng (ứng dụng với hàm scanf). 134](#_Toc126327138)

[VD13 Truy suất phần tử trong mảng 135](#_Toc126327139)

[VD14: Truyền mảng vào hàm. 136](#_Toc126327140)

[Mảng con trỏ 138](#_Toc126327141)

[Phân biệt con trỏ mảng và mảng con trỏ 138](#_Toc126327142)

[Con trỏ chuỗi 139](#_Toc126327143)

[Con trỏ của con trỏ 140](#_Toc126327144)

[Con trỏ mảng 2 chiều 141](#_Toc126327145)

[a) Khai báo mảng 2 chiều trong C dưới dạng con trỏ trong mảng 2 chiều 142](#_Toc126327146)

[b) Khai báo mảng 2 chiều trong C dưới dạng con trỏ trong mảng 1 chiều 143](#_Toc126327147)

[c) Sử dụng con trỏ mảng 2 chiều trong hàm 145](#_Toc126327148)

[d) Sử dụng con trỏ mảng 2 chiều trong struct 146](#_Toc126327149)

[e) Thêm hàng, cột, xoá hàng, cột. 147](#_Toc126327150)

[Tạo thư viện 147](#_Toc126327151)

[Tạo thư viện với vscode: 147](#_Toc126327152)

[Chú ý: 147](#_Toc126327153)

[VD1: 148](#_Toc126327154)

[VD2: 149](#_Toc126327155)

[Hàm (function) 150](#_Toc126327156)

[Hàm memcpy() 151](#_Toc126327157)

[Nguyên mẫu hàm 151](#_Toc126327158)

[Khái niệm: 151](#_Toc126327159)

[Cách khai báo: 151](#_Toc126327160)

[Vị trí đặt của nguyên mẫu hàm: 152](#_Toc126327161)

[VD1: 152](#_Toc126327162)

[VD2: 153](#_Toc126327163)

[#ifndef, #ifdef, #define, #endif 154](#_Toc126327164)

[Tham khảo 154](#_Toc126327165)

# 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

Thám khảo: <http://laptrinhcanban.com/c/lap-trinh-c-co-ban/ham-trong-c/void-trong-c/>

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 |

### Hàm Strchr(): Tìm ký tự trong chuỗi

Hàm strchr() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **tìm kiếm ký tự chỉ định trong chuỗi C**.

Cú pháp

strchr(str, chr);

Trong đó:

* str là chuỗi ban đầu
* chr là ký tự cần tìm trong chuỗi str.

Hàm strchr() sẽ trả về vị trí đầu tiên tìm thấy của ký tự chr trong chuỗi str tính từ đầu chuỗi, và trả về vị trí này dưới dạng con trỏ chuỗi trong c.

VD25

#include <stdio.h>

#include <string.h>

int main(void){

    char str[30] = "Good School";

    //Khai báo con trỏ chuỗi để chứa kết quả

    char \* p;

    p = strchr(str, 'o');

    printf("%s\n", p);

}

Kết quả:

ood School

### Hàm Strstr: tìm chuỗi con trong chuỗi

Hàm strstr() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **tìm kiếm chuỗi con trong chuỗi C**. Kết quả trả về sẽ là địa chỉ của vị trí đầu tiên tìm thấy chuỗi con trong chuỗi ban đầu dưới dạng con trỏ chuỗi trong c.

Cú pháp:

strstr(str, keyword);

Trong đó:

* str là chuỗi ban đầu
* keyword là chuỗi ký tự cần tìm trong chuỗi str.

Hàm strstr() sẽ trả về địa chỉ của vị trí đầu tiên tìm thấy của chuỗi ký tự keyword trong chuỗi str tính từ đầu chuỗi, và trả về vị trí này dưới dạng con trỏ chuỗi trong c.

VD02:

#include <stdio.h>

#include <string.h>

int main(void){

    char str[30] = "Good School";

    char \* p;

    p = strstr(str, "oo");

    printf("%s\n", p);

}

Kết quả:

ood School

### Tìm vị trí ký tự trong chuỗi

Chúng ta có thể dùng tới vòng lặp for để lấy và so sánh từng ký tự trong chuỗi ban đầu với ký tự cần tìm để tìm ra vị trí của ký tự đó trong chuỗi ban đầu.

VD26:

#include <stdio.h>

#include <string.h>

int main(void){

  char str[]= "Good School";

  char chr = 'o';

  //Tính độ dài chuỗi ban đầu

  int len = strlen(str);

  //sử dụng vòng lặp for để tìm vị trí ký tự "o" trong chuỗi

  for (int i=0; i < len; i++){

      //so sánh ký tự có index i với "o"

      if (str[i] == chr) printf("Tim thay ky tu %c tai vi tri: %d\n",chr,i);

  }

  return 0;

}

Tim thay ky tu o tai vi tri: 1

Tim thay ky tu o tai vi tri: 2

Tim thay ky tu o tai vi tri: 8

Tim thay ky tu o tai vi tri: 9

### Tìm số lần xuất hiện của một ký tự trong một chuỗi

Để tìm số lần xuất hiện của một ký tự trong một chuỗi trong C, chúng ta có thể sử dụng lại cách tìm vị trí ký tự trong chuỗi C ở trên. Tuy nhiên chúng ta cũng cần thêm xử lý đếm số lần xuất hiện vào chương trình.

VD27

#include <stdio.h>

#include <string.h>

int main(void){

  char str[]= "Good School";

  char chr = 'o';

  int count= 0;

  //Tính độ dài chuỗi ban đầu

  int len = strlen(str);

  //sử dụng vòng lặp for để tìm vị trí ký tự "o" trong chuỗi

  for (int i=0; i < len; i++){

      //so sánh ký tự có index i với "o"

      if (str[i] == chr){

          count += 1;

      }

  }

  printf("Tong so lan tim thay ky tu: %d",count);

}

Tong so lan tim thay ky tu: 4

### So sánh 2 ký tự char

Khi so sánh lớn nhỏ hai ký tự đơn, chúng ta đơn giản so sánh mã ký tự ASSCII của chúng. Ví dụ, do điểm mã unicode của ký tự a là 97 sẽ lớn hơn của ký tự A à 65 nên phép so sánh sau sẽ cho ra kết quả true:

'a' > 'A'

// true

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 | 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.

Chúng ta sử dụng hàm strcmp() để so sánh 2 chuỗi trong C với cú pháp sau đây:

strcmp(str1, str2);

Trong đó str1 và str2 là 2 chuỗi cần được so sánh.

Hàm strcmp() sẽ tiến hành so sánh bắt đầu từ ký tự đầu tiên trong hai chuỗi, và nếu chúng giống nhau, các ký tự tiếp theo sẽ lần lượt được so sánh cho tới khi xuất hiện cặp 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.

Sau khi đã xác định được cặp **ký tự đầu tiên khác nhau giữa 2 chuỗi**, hàm strcmp() sẽ trả về **hiệu mã ký tự ASSCII giữa chúng**, và bằng cách so sánh giá trị này với số 0, chúng ta có thể so sánh 2 chuỗi như sau:

| **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 |

VD28

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main(void)

{

  char s1[] = "ab";

  char s2[] = "aAb";

  printf("Ma ascii cua a:%d\n", 'a');

  printf("Ma ascii cua A:%d\n", 'A');

  printf("Ma ascii cua b:%d\n", 'b');

  printf("Hieu ma ascii: %d\n",strcmp(s1,s2));

  if(strcmp(s1,s2)==0){

     printf("%s bang %s", s1,s2);

  } else if(strcmp(s1,s2)>0){

         printf("%s lon hon %s", s1,s2);

  } else

        printf("%s nho hon %s", s1,s2);

    return 0;

}

Ma ascii cua a:97

Ma ascii cua A:65

Ma ascii cua b:98

Hieu ma ascii: 1

ab lon hon aAb

### So sánh 2 chuỗi trong C với số ký tự chỉ định| Hàm strncmp()

Trong trường hợp bạn chỉ muốn so sánh một số ký tự chỉ định giữa 2 chuỗi, ví dụ như là chỉ so sánh 3 ký tự đầu tiên trong mỗi chuỗi ban đầu chẳng hạn, khi đó chúng ta sẽ dùng tới Hàm strncmp().

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**.

Chúng ta sử dụng hàm strncmp() để so sánh 2 chuỗi trong C với số ký tự chỉ định với cú pháp sau đây:

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.

Hàm strncmp() cũng sử dụng hiệu mã ký tự ASSCII giữa hai chuỗi để so sánh chúng. Điểm khác biệt duy nhất đó là thay vì so sánh trên toàn bộ 2 chuỗi ban đầu, thì chúng ta chỉ tiến hành so sánh trên một phạm vi được quyết định bởi đối số n khi dùng hàm strcmp() mà thôi.

VD29

#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;

}

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

### Hàm isspace() | Kiểm tra một ký tự có phải là khoảng trắng hay không

Hàm isspace() trong C là một hàm có sẵn header file ctype.h, giúp chúng ta kiểm tra một ký tự có phải là khoảng trắng hay không.

Cú pháp

isspace(c);

Trong đó c là ký tự cần được kiểm tra.

Hàm isspace() sẽ trả về 1 nếu như ký tự được chỉ định là khoảng trắng, và trả về 0 trong các trường hợp còn lại.

Các giá trị được coi là khoảng trắng trong C như bảng sau, và nếu ký tự được chỉ định thuộc một trong các giá trị này, thì hàm isspace() sẽ trả về 1.

| **Ký hiệu** | **Code** | **Ý nghĩa** |
| --- | --- | --- |
| \t | 0x09 | tab ngang (TAB) |
| \n | 0x0A | dòng mới(LF) |
| \v | 0x0B | tab doc(VT) |
| \f | 0x0C | Ngắt trang(FF) |
| \r | 0x0D | Hồi quy(CR) |
| ‘ ‘ | (0x20) | khoảng trống (SPACE) |

VD30

#include<stdio.h>

#include<ctype.h>

int main(){

    int c[5] = {'\n', '2', 1, ' ', ','};

    for(int i = 0; i < 5; i++){

        if(isspace(c[i])) {

                printf("c[%d] la khoang trang\n",i);

        }

    }

}

c[0] la khoang trang

c[3] la khoang trang

### Đếm khoảng trắng

VD31

#include<stdio.h>

#include<ctype.h>

#include <string.h>

int main(void){

  int n = 0;

  char str[100] = "27 thang 8\t2021";

  printf("%s\n",str);

  const int len = strlen(str);

  for(int i = 0; i < len; i++){

    if(isspace(str[i])) n += 1;

  }

  printf("So khoang trang: %d\n",n);

  return 0;

}

27 thang 8 2021

So khoang trang: 3

### Xóa toàn bộ ký tự trắng

VD32

#include <stdio.h>

#include <string.h>

char\* delete\_space(char \*str,char \*result) {

  int pos = 0;

  const int len = strlen(str); /\*Tìm độ dài chuỗi ban đầu\*/

  for(int i = 0; i < len; i++) {

      char c = str[i];

      /\*Nếu ký tự lấy ra là ký tự trắng thì bỏ qua\*/

      if (c == '\r' || c == '\n' || c == ' ') {

        continue;

      }

      /\*Thêm ký tự lấy ra vào kết quả\*/

      result[pos++] = c;

  }

  return result;

}

int main(void) {

    char str[] = "   Hello World     ";

    printf("[%s]\n", str);

    char result[100];

    printf("[%s]\n",(delete\_space(str,result)));

}

[ Hello World ]

[HelloWorld]

### Xóa khoảng trắng đầu và cuối chuỗi

VD33

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

char \*trimString(char \*str){

    char \*end; /\*Khai con trỏ của ký tự cuối cùng\*/

    /\*Vòng lặp đầu tiên để xóa khoảng trắng từ đầu chuỗi\*/

    while(isspace((unsigned char)\*str)) str++;

    /\*Nếu vòng lặp thực hiện hết cả chuỗi thì trả về chuỗi str\*/

    if(\*str == 0) return str;

    /\*end là vị trí cuối chuỗi, cũng là vị trí bắt đầu vòng lặp thứ 2\*/

    end = str + strlen(str) - 1;

    /\*Vòng lặp thứ hai để xóa khoảng trắng từ cuối chuỗi\*/

    while(end > str && isspace((unsigned char)\*end)) end--;

    end[1] = '\0';

    return str;

}

int main(void) {

    const char \*str1 = "   Hello World     ";

    printf("[%s]\n", str1);

    /\*Dùng hàm strdup để copy chuỗi ban đầu và trả về con trỏ của chuỗi đó\*/

    /\*Gán con trỏ ở tạo bởi strdup vào con trỏ tmp\*/

    char \*tmp = strdup(str1);

    printf("[%s]\n", trimString(tmp));

    /\*Trách nhiệm giải phóng bộ nhớ thuộc về bên gọi\*/

    free(tmp);

    exit(EXIT\_SUCCESS);

}

[ Hello World ]

[Hello World]

### Hàm strcpy

Hàm strcpy() trong C là một hàm có sẵn trong header file string.h, giúp chúng ta copy toàn bộ chuỗi chỉ định (bao gồm cả ký tự kết thúc chuỗi \0).

Cú pháp:

strcpy(target, source);

Trong đó:

* targe là chuỗi đích
* source là chuỗi nguồn.

VD03:

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[30];

    char str2[30] = "I love C/C++";

    //Copy toàn bộ chuỗi str2 vào chuỗi str1

    strcpy(str1, str2);

    printf("%s\n", str1);

}

Kết quả:

I love C/C++

### Độ dài chuỗi:

Chúng ta coi độ dài chuỗi trong C chính là tổng số mã ký tự tạo nên các ký tự trong nó.

Ví dụ chuỗi bigcityboy được tạo bởi 10 ký tự, nên độ dài của chuỗi này bằng 10 đơn vị.

Lưu ý rằng độ dài của chuỗi có thể khác với số ký tự có trong chuỗi, trong trường hợp tồn tại ký tự được tạo bởi hơn 2 mã ký tự, như là ký tự tiếng Việt có dấu, hoặc là kanji tiếng Nhật chẳng hạn.

char srt1[]="bigcityboy";

// độ dài chuỗi str1 = 10

char srt2[]="đế";

// độ dài chuỗi str2 = 5

Tính độ dài chuỗi bằng hàm strlen trong C

**Hàm strlen trong C** là một hàm có sẵn header file **string.h**, giúp chúng ta **tính độ dài chuỗi** trong đối số.

Cú pháp:

strlen(str);

Hàm strlen() sẽ trả về độ dài chuỗi ký tự được gán trong chuỗi. Lưu ý rằng đây là độ dài (số ký tự) của chuỗi giá trị gán trong chuỗi, do đó nó sẽ **không bao gồm** cả ký tự kết thúc chuỗi \0 khi tính độ dài.

#include <stdio.h>

#include <string.h>

int main(void)

{

char str[] = "I Love C/C++";

int len;

len = strlen(str);

printf("Length of \"%s\" is %d.\n", str, len);

}

//Length of "I Love C/C++" is 12.

### Con trỏ chuỗi

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

Giống con trỏ mảng.

Khai báo: char \*p;

char a[] = "Hello", \*p1;

p1 = a;

char \*array\_str ="Hello";

Truy xuất: \*(p + index);

### Chèn chuỗi

Việc chèn một chuỗi sẽ làm tăng tổng số ký tự của chuỗi ban đầu. Do đó, nếu bạn muốn chèn một chuỗi khác vào chuỗi ban đầu, hãy đảm bảo là bộ nhớ cấp phát cho chuỗi ban đầu đủ khả năng để chứa hết chuỗi cần chèn. Nếu không thì sự cố tràn bộ nhớ sẽ bị xảy ra.

/\* Chuỗi ban đầu chỉ có tối đa 7 ký tự và không thể chèn thêm \*/

char str1[] = "abcde";

/\* Chuỗi ban đầu có tối đa 10 ký tự và có thể chèn thêm chuỗi "xyz" \*/

char str2[10] = "abcde";

Cách 1

Các bước tiến hành chèn chuỗi vào chuỗi lúc này như sau:

1. Bắt đầu từ vị trí cần chèn trong chuỗi ban đầu, chúng ta dịch chuyển dần về phía sau lượng ký tự bằng đúng với số ký tự của chuỗi cần chèn và tạo ra các ô trống.
2. Sao chép chuỗi ký tự cần chèn để chèn vào các ô trống vừa tạo.

VD4

#include <assert.h>

#include <stdio.h>

#include <string.h>

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*

    Chèn chuỗi trong C

    Tham số

        str1:      Chuỗi ban đầu (không chấp nhận Null)

        str1\_size: Độ dài chuỗi str1

        pos:       Vị trí chèn (tính từ đầu chuỗi str1)

        str2:      Chuỗi chèn vào (không chấp nhận Null pointer)

    Giá trị trả về

        str1

\*/

char\* str\_insert(char\* str1, size\_t str1\_size, size\_t pos, const char\* str2)

{

    assert(str1 != NULL);

    assert(str2 != NULL);

    size\_t len1 = strlen(str1);

    size\_t len2 = strlen(str2);

    // Kiểm tra str1 có đủ độ dài để chèn str2 không

    assert(len1 + len2 < str1\_size);

    // Kiểm tra vị trí chèn có nằm trong chuỗi str1 không

    assert(len1 >= pos);

    // Làm trống một phạm vi dài với len2 ký tự, từ  str1[pos])

    // Dịch chuyển chuỗi ban đầu về sau, bao gồm cả ký tự `\0'

    memmove(&str1[pos + len2], &str1[pos], len1 - pos + 1);

    // Copy str2 và dán vào khoảng trống mới tạo

    memcpy(&str1[pos], str2, len2);

    return str1;

}

int main(void)

{

    char str[10] = "abcde";

    printf("before: %s\n",str);

    printf("after: %s",str\_insert(str, SIZE\_OF\_ARRAY(str), 2, "xyz"));

    return 0;

}

Kết quả:

before: abcde

after: abxyzcde

Cách 2: cấp phát bộ nhớ động khi chèn chuỗi

Hàm str\_alloc\_and\_insert() chỉ sử dụng nội dung của chuỗi ban đầu mà không làm thay đổi nó. Bởi vậy , chúng ta sử dụng tới const char\* str1 để biểu thị ý nghĩa này.

Do chúng ta đã cấp phát bộ nhớ động trong hàm, nên sẽ không phải lo lắng chuỗi kết quả có đủ bộ nhớ để chèn hay không, và sự cố **tràn bộ nhớ** tất nhiên sẽ không xảy ra.

Kết quả trả về của hàm này không phải là chuỗi kết quả, mà là một con trỏ trỏ đến vùng được cấp phát động để lưu giữ kết quả chèn chuỗi trong hàm.

Chú ý là hàm free() cần được gọi bên ngoài hàm str\_alloc\_and\_insert() để có thể giải phóng bộ nhớ đã cấp phát sau khi kết thúc hàm.

Trong hàm dưới chúng ta đã sử dụng hàm malloc() để cấp phát bộ nhớ động, và tiến hành copy 3 lần chuỗi bằng hàm memcpy(), là phần đầu chuỗi ban đầu, chuỗi cần chèn, và phần cuối chuỗi ban đầu. Ký hiệu kết thúc chuỗi \0 cũng cần được thêm vào để hoàn thành chuỗi.

VD5

#include <assert.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*

    Chèn chuỗi trong C

    Tham số

        str1:      Chuỗi ban đầu (không chấp nhận Null)

        pos:       Vị trí chèn (tính từ đầu chuỗi str1)

        str2:      Chuỗi chèn vào (không chấp nhận Null pointer)

    Giá trị trả về

        Con trỏ mảng chứa kết quả. Nếu thực thi thất bại thì trả về Null

\*/

char\* str\_alloc\_and\_insert(const char\* str1, size\_t pos, const char\* str2)

{

    assert(str1 != NULL);

    assert(str2 != NULL);

    size\_t len1 = strlen(str1);

    size\_t len2 = strlen(str2);

    // Kiểm tra vị trí chèn có nằm trong chuỗi str1 không

    assert(len1 >= pos);

    size\_t result\_size = len1 + len2 + 1;

    char\* result = malloc(sizeof(char) \* result\_size);

    if (result == NULL) {

        return NULL;

    }

    // Copy phạm vi từ đầu chuỗi ban đầu tới trước vị trí cần chèn vào chuỗi kết quả

    memcpy(result, str1, pos);

    // Copy chuỗi cần chèn vào cuối kết quả

    memcpy(&result[pos], str2, len2);

    // Copy phạm vi từ vị trí cần chèn tới cuối chuỗi ban đầu vào chuỗi kết quả

    // Thêm ký tự kết thúc chuỗi `\0` vào kết quả

    strcpy(&result[pos + len2], &str1[pos]);

    return result;

}

int main(void)

{

    char str[] = "abcde";

    printf("before: %s\n",str);

    char\* result = str\_alloc\_and\_insert(str, 2, "xyz");

    if (result != NULL) {

         printf("after: %s\n",result);

        free(result);

    }

    return 0;

}

Kết quả:

before: abcde

after: abxyzcde

### Copy chuỗi

Chúng ta có 3 phương pháp để copy chuỗi trong C như sau:

* Hàm strcpy(): Copy toàn bộ chuỗi ban đầu
* Hàm strcpy\_s(): Copy toàn bộ chuỗi ban đầu
* Hàm strncpy(): Copy số lượng ký tự chỉ định từ chuỗi ban đầu

Copy toàn bộ chuỗi trong C | hàm strcpy()

Hàm strcpy() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **copy toàn bộ chuỗi chỉ định** (bao gồm cả ký tự kết thúc chuỗi \0).

Cú pháp:

strcpy(target, source);

Hàm strcpy() sẽ copy toàn bộ nội dung của chuỗi source và phục chế nó vào chuỗi target. Lưu ý là hai chuỗi này cần phải tồn tại trước đó, thông qua việc [khởi tạo hay khai báo chuỗi trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/chuoi-string-trong-c/khai-bao-chuoi-trong-c).

VD6

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[30];

    char str2[30] = "I love C/C++";

    //Copy toàn bộ chuỗi str2 vào chuỗi str1

    strcpy(str1, str2);

    printf("%s\n", str1);

}

Kết quả:

I love C/C++

Lưu ý về hiện tượng tràn bộ nhớ (Buffer overflow). Yêu cầu biến đích phải đủ bộ nhớ để lưu chuỗi.

Tham khảo: <http://laptrinhcanban.com/c/lap-trinh-c-co-ban/chuoi-string-trong-c/copy-chuoi-trong-c/>

Copy toàn bộ chuỗi trong C | hàm strcpy\_c()

Nếu bạn đang phát triển chương trình C trên môi trường Visual Studio, chúng ta không thể sử dụng tới hàm copy chuỗi strcpy() được.

Khi tiến hành compile chương trình chứa hàm strcpy() thì lỗi compile sẽ xảy ra, và có thêm màn hình hướng dẫn chúng ta hãy sử dụng tới hàm strcpy\_c() để thay thế.

Lý do cũng dễ hiểu, bởi việc copy chuỗi bằng hàm strcpy() có một nhược điểm là nếu kích thước chuỗi cần copy lớn hơn kích thước của chuỗi đích chứa nó, thì hiện tượng **tràn bộ nhớ** sẽ bị xảy ra.

Và để khắc phục điểm yếu này của hàm strcpy() mà hàm strcpy\_c() đã được ra đời. Trong hàm strcpy\_c() có thêm chỉ định kích thước chuỗi đích, giúp chúng ta quản lý và không để xảy ra hiện tượng **tràn bộ nhớ** trong quá trình copy chuỗi trong C.

Lưu ý rằng **strcpy\_c() không phải là một hàm chuẩn trong C**, và nó có thể **không hoạt động** trong một số môi trường lập trình C.

Chúng ta sử dụng hàm strcpy\_c() để copy toàn bộ chuỗi trong C với cú pháp sau đây:

strcpy\_c(target, rsize\_t, source);

Trong đó:

* target là chuỗi đích
* rsize\_t là kích thước chuỗi đích
* source là chuỗi nguồn.

Lưu ý là chúng ta cũng cần phải viết thêm dòng #include <string.h> để load header file **string.h** để có thể sử dụng được hàm strcpy\_c() trong chuơng trình.

VD7

#include <stdio.h>

#include <string.h>

int main(void) {

    char str1[16] = "Hello World!";

    char str2[16];

    //Copy toàn bộ chuỗi str2 vào chuỗi str1

    strcpy\_s(str2, 16, str1);

    printf("%s\n", str2);

    return 0;

}

Kết quả:

Hello World!

Copy số lượng ký tự chỉ định từ chuỗi ban đầu | hàm strncpy()

Hàm strncpy() trong C cũng là một hàm có sẵn trong header file **string.h**, giúp chúng ta **Copy số lượng ký tự chỉ định từ chuỗi ban đầu**.

Chúng ta sử dụng hàm strncpy() để copy toàn bộ chuỗi trong C với cú pháp sau đây:

strncpy(target, source, n);

Trong đó:

* target là chuỗi đích
* source là chuỗi nguồn.
* n là số ký tự lớn nhất có thể copy

Hàm strncpy() sẽ copy từ chuỗi source một số lượng n ký tự, và dán nó vào chuỗi target. Hai chuỗi này cần phải tồn tại trước đó, thông qua việc [khởi tạo hay khai báo chuỗi trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/chuoi-string-trong-c/khai-bao-chuoi-trong-c).

Cần chú ý là nếu chỉ định **giá trị của n nhỏ hơn độ dài của chuỗi nguồn** khi sử dụng hàm strncpy() thì ký tự kết thúc chuỗi \0 sẽ không được copy vào chuỗi đích, do đó chúng ta cần phải thêm xử lý nối ký tự này vào chuỗi đích target.

Và chúng ta cần phải viết thêm dòng #include <string.h> để load header file **string.h** để có thể sử dụng được hàm strlen() trong chuơng trình.

VD8

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[30];

    char str2[30] = "I love C/C++";

    //Copy nhiều nhất 6 ký tự từ chuỗi str2 vào chuỗi str1

    strncpy(str1, str2,6);

    str1[6] = '\0'; //Gán ký tự kết thúc chuỗi \0

    printf("%s\n", str1);

}

Kết quả:

I love

### Nối chuỗi

Chúng ta có 3 phương pháp để nối chuỗi trong C như sau:

* Hàm strcat(): nối 2 chuỗi trong c
* Hàm strncat(): nối 2 chuỗi trong c với số lượng ký tự chỉ định
* Hàm sprintf(): nối nhiều chuỗi trong c

Nối 2 chuỗi trong C | Hàm strcat()

Hàm strcat() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **nối 2 chuỗi trong c** lại với nhau.

Chúng ta sử dụng hàm strcat() để nối chuỗi trong C với cú pháp sau đây:

strcat(target, source);

Trong đó:

* target là chuỗi đích
* source là chuỗi dùng để nối vào chuỗi target

Hàm strcat() sẽ copy toàn bộ nội dung của chuỗi source, sau đó gán vào vị trí cuối cùng của chuỗi target và trả về kết quả là con trỏ của chuỗi kết quả.

VD9

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[20] = "I Lov";

    char str2[10] = "e You";

    strcat(str1, str2);

    printf("%s\n", str1);

}

Kết quả:

I Love You

Nối 2 chuỗi trong C với số lượng ký tự chỉ định| Hàm strncat()

Hàm strncat() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **nối 2 chuỗi trong c** lại với nhau với số lượng ký tự chỉ định.

Chúng ta sử dụng hàm strncat() để nối chuỗi trong C với cú pháp sau đây:

strncat(target, source, n);

Trong đó:

* target là chuỗi đích
* source là chuỗi dùng để nối
* n là số ký tự tối đa được lấy ra từ đầu chuỗi source để nối vào chuỗi target

Hàm strncat() sẽ copy n ký tự tính từ đầu chuỗi source, sau đó gán chúng vào vị trí cuối cùng của chuỗi target và trả về kết quả là con trỏ của chuỗi kết quả.

VD10

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[20] = "I Lov";

    char str2[10] = "e You";

    strncat(str1, str2,3);

    printf("%s\n", str1);

}

Kết quả:

I Love Y

Nối nhiều chuỗi trong c | Hàm sprintf()

Hàm sprintf() trong C vốn là một hàm có sẵn trong header file **stdio.h**, giúp chúng ta tạo **tạo một chuỗi đầu ra từ các biến** với kiểu và định dạng khác nhau.

* Xem thêm: [Bốn cách xuất dữ liệu trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/nhap-xuat-trong-c/bon-cach-xuat-du-lieu-trong-c)

Trong trường hợp chỉ định các biến này là các chuỗi, chúng ta có thể **nối nhiều chuỗi trong c** lại và tạo thành một chuỗi mới thông qua hàm sprintf().

Cú pháp của hàm sprintf() để nối nhiều chuỗi trong C như sau:

sprintf (target, \*format, str1 ,str2 , ...);

Trong đó

* target là chuỗi đích
* str là các chuỗi dùng để nối vào chuỗi target
* \*format là chuỗi định dạng, là tập hợp các định dạng %s của các chuỗi str. Có bao nhiêu chuỗi str được chỉ định 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).

VD11

#include <stdio.h>

int main(void) {

    char str1[10] = "Hello";

    char str2[10] = "C";

    char str3[10] = "World!";

    char target[100];

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

    printf("%s", target);

    return 0;

}

Kết quả:

Hello C World!

### Tách chuỗi | strtok()

Hàm strtok() trong C là một hàm có sẵn trong header file **string.h**, giúp chúng ta **tách chuỗi trong chuỗi C** bằng **ký tự chỉ định**.

Để có thể sử dụng được hàm strtok(), chúng ta cần phải viết thêm dòng #include <string.h> để load header file **string.h** vào trong chuơng trình.

Chúng ta sử dụng hàm strtok() để tách chuỗi trong chuỗi C với cú pháp sau đây:

strtok(str, sep);

Trong đó:

* str là chuỗi cần tách
* sep là ký tự phân tách. (sep viết tắt separator)

Hàm strtok() sẽ trả về **vị trí đầu chuỗi** của **chuỗi con đầu tiên được phân tách bằng dấu phân cách** từ chuỗi ban đầu dưới dạng con trỏ chuỗi trong C. Trong trường hợp không tìm thấy **ký tự phân tách** , giá trị NULL sẽ được trả về.

Cơ chế của hàm strtok() là **điền ký tự kết thúc chuỗi \0 vào chuỗi ban đầu** mỗi khi tìm thấy *ký tự phân tách*, do đó cần lưu ý là hàm strtok() sẽ **làm biến đổi** chuỗi ban đầu.

Lại nữa, hàm strtok() chỉ có thể tách một chuỗi con bằng dấu phân tách từ chuỗi ban đầu trong mỗi lần thực thi mà thôi.

VD12

#include <stdio.h>

#include <string.h>

int main(void){

    char str[50] = "Ha Noi,Ninh Binh,Nam Dinh,Thanh Hoa";

    //Khai báo con trỏ chuỗi để chứa kết quả

    char \* p;

    p = strtok(str, ",");

    printf("%s\n", p);

}

Kết quả:

Ha Noi

Ở ví dụ này, hàm strtok() sẽ hoạt động với các bước như sau:

1. Hàm strtok() bắt đầu tìm vị trí ký tự phân tách là dấu , ở bên trong chuỗi str.
2. Sau khi tìm thấy vị trí dấu phẩy đầu tiên, hàm strtok() sẽ điền ký tự kết thúc chuỗi \0 vào vị trí tìm thấy. Khi đó, chuỗi ban đầu sẽ chuyển thành dạng Ha Noi\0Ninh Binh,Nam Dinh,Thanh Hoa.
3. Hàm strtok() quay trở về đầu chuỗi ban đầu và trả về vị trí đầu chuỗi này dưới dạng con trỏ chuỗi trong C. Cuối cùng khi in con trỏ chuỗi này, chuỗi sẽ được in ra từ vị trí đầu chuỗi là ký tự H cho đến ký tự thúc chuỗi \0 (Ha Noi\0), do đó chuỗi kết quả Ha Noi sẽ được in ra màn hình.

Tách toàn bộ chuỗi con từ chuỗi trong C | hàm strtok

Để có thể tách toàn bộ chuỗi con từ chuỗi ban đầu trong C thì sau lần tách đầu tiên, chúng ta cần phải tạo ra một vòng lặp để **lần lượt tách** các chuỗi con con lại từ chuỗi ban đầu bằng hàm strtok().

VD13

#include <stdio.h>

#include <string.h>

int main(void){

    char str[50] = "Ha Noi,Ninh Binh,Nam Dinh,Thanh Hoa";

    char \* p;

    //Tách chuỗi con lần đầu tiên

    p = strtok(str, ",");

    printf("%s\n", p);

    //Tách chuỗi con từ lần thứ 2 trở đi

    //Bằng cách sử dụng hàm strok cho tới khi kết quả NULL được trả về.

    while(p != NULL) {

        //Chỉ dịnh đối số NULL trong hàm strtok để tiếp tục tách chuỗi ban đầu

        p = strtok(NULL, ",");

        if(p != NULL) {

            printf("%s\n", p);

        }

    }

    return 0;

}

Kết quả:

Ha Noi

Ninh Binh

Nam Dinh

Thanh Hoa

VD16

#include <stdio.h>

#include <string.h>

int main(void){

    char str[50] = "Tom and Jerry andmae";

    char \* p;

    p = strtok(str, " and ");

    printf("%s\n", p);

    while(p != NULL) {

        p = strtok(NULL, " and ");

        if(p != NULL) {

            printf("%s\n", p);

        }

    }

    return 0;

}

Kết quả:

Tom

Jerry

m

e

### Tách số trong chuỗi

Chúng ta có 2 phương pháp để tách số trong chuỗi C như sau:

* Hàm isdigit(): tách số trong chuỗi C bằng cách kiểm tra từng ký tự
* Tách số trong chuỗi C bằng hàm strtok()

Tách số trong chuỗi C bằng cách kiểm tra từng ký tự | Hàm isdigit()

Hàm isdigit() vốn được sử dụng để kiểm tra chuỗi có phải la số hay không trong C. Ứng dụng hàm này vào trong một vòng lặp, chúng ta có thể tiến hành kiểm tra từng ký tự trong chuỗi có phải là số hay không và tách ra nếu đó là số.

VD14

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int main(){

    char str[] = "A100B20C3000D40";

    char result[100];

    int i=0;

    int n=0;

    for(i=0; i<=strlen(str); ++i){

        /\* Kiểm tra ký tự thứ i trong chuỗi đã cho có phải là số hay không\*/

        /\* Nếu là số thì lưu vào chuỗi result\*/

        if(isdigit(str[i])){

            result[n] = str[i];

            n++;

            result[n] = '\0';

        }else if(n){ /\* Nếu chuỗi result chứa phần tử thì in chuỗi result\*/

            printf("%s\n",result);

            n=0; /\* Reset và làm rỗng chuỗi result\*/

        }

    }

    return 0;

}

Kết quả:

100

20

3000

40

Tách số trong chuỗi C bằng hàm strtok()

Hàm strtok() vốn được sử dụng để tách chuỗi trong chuỗi C bằng ký tự chỉ định. Ứng dụng hàm strtok() và chỉ định ký tự tách chuỗi là dãy chữ cái không phải là số, chúng ta có thể thực hiện việc tách số trong chuỗi C.

VD15

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int main(void){

    char str[] = "A100B20C3000D40";

    char \*p, \*q;

    p=str;

    while(q=strtok(p,"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz")){

        p=NULL;

        printf("%s\n", q);

    }

    return 0;

}

Kết quả:

100

20

3000

40

Lưu ý: cách này không nên dùng, do nếu trong chuỗi ban đầu có kí tự đặc biệt, thì kết quả sẽ không tách được kí tự đặc biệt ra.

### Cắt chuỗi

Cắt chuỗi trong C bằng hàm strncpy()

Hàm strncpy() trong C vốn được sử dụng giúp chúng ta **copy số lượng ký tự chỉ định từ chuỗi ban đầu**. Ứng dụng hàm strncpy(), chúng ta có thể copy một số lượng ký tự từ trong phạm vi chuỗi ban đầu để tạo ra chuỗi con, và qua đó hoàn thành được việc cắt chuỗi trong C.

strncpy(target, source + begin, n)

Trong đó:

* target là chuỗi đích để chứa kết quả cắt
* source là chuỗi nguồn.
* begin là vị trí bắt đầu cắt trong chuỗi nguồn
* n là số ký tự sẽ cắt từ vị trí begin.

Hiểu đơn giản thì cú pháp trên sẽ **cắt n ký tự từ vị trí begin trong chuỗi source và trả về kết quả là chuỗi target**.

Cần chú ý là nếu chỉ định **giá trị của n nhỏ hơn độ dài của chuỗi nguồn** khi sử dụng hàm strncpy() thì ký tự kết thúc chuỗi \0 sẽ không được copy vào chuỗi đích, do đó chúng ta cần phải thêm xử lý nối ký tự này vào chuỗi đích target.

VD17

#include <stdio.h>

#include <string.h>

void main() {

    char str[] = "01234567890";

    char t[64];

    strncpy( t, str+3, 5 ); /\*Cắt 5 ký tự từ vị trí thứ 3 \*/

    t[5] = '\0';            /\*Thêm ký tự kết thúc chuỗi vào kết quả\*/

    printf( "%s\n", t );

}

34567

VD18

#include <assert.h>

#include <stdio.h>

#include <string.h>

/\*Tạo hàm cắt chuỗi trong C\*/

char\* substring(char\* source, size\_t begin, size\_t n, char\* target){

    int target\_size = sizeof(target);

    /\*Kiểm tra đầu vào và phát hiện lỗi nếu có\*/

    assert(source != NULL);

    assert(target != NULL);

    assert(n + 1 <= target\_size);

    /\*Kiểm tra vị trí cắt có nằm trong chuỗi không\*/

    if (begin < strlen(source)) {

        /\*Cắt n ký tự từ vị trí begin trong chuỗi source\*/

        strncpy(target, source + begin, n);

        /\*Thêm ký tự kết thúc chuỗi vào kết quả\*/

        target[n] = '\0';

    }

    /\* Bỏ qua hàm nếu như vị trí bắt đầu cắt nằm ngoài chuỗi ban đầu\*/

    else {

        target[0] = '\0';

    }

    return target;

}

int main(void){

    char str[] = "abcdef";

    char target[4];

    puts(substring(str, 0, 3, target )); //Cắt 3 ký tự từ vị trí 0

    puts(substring(str, 2, 3, target )); //Cắt 3 ký tự từ vị trí 2

    puts(substring(str, 2, 0, target));  //Cắt 0 ký tự từ vị trí 2

    puts(substring(str, 5, 3, target));  //Cắt 3 ký tự từ vị trí 5

    return 0;

}

Kết quả:

abc

cde

f

### Xóa ký tự trong chuỗi

Để xóa 1 ký tự trong chuỗi C cũng khá đơn giản, chúng ta tạo ra một vòng lặp để lấy ra từng ký tự trong chuỗi ban đầu, và kiểm tra xem ký tự đó có giống với ký tự chỉ định cần xóa hay không.

Lúc này sẽ có 2 cách làm, một là kiểm tra từng ký tự trong chuỗi ban đầu và xóa đi nếu giống với ký tự cần xóa, và hai là kiểm tra và lấy các ký tự không giống ký tự cần xóa từ chuỗi ban đầu.

VD19

#include<stdio.h>

char\* pop\_str(char\* a, char chr, char\* b){

  int a\_cnt;

  int b\_cnt = 0;

  /\* So sánh từng ký tự trong chuỗi a với ký tự c\*/

  for ( a\_cnt = 0; a[a\_cnt] != '\0'; ++a\_cnt ) {

    /\* Nếu ký tự khác c thì lấy ra và thêm vào kết quả \*/

    if ( a[a\_cnt] != chr ) {

      b[b\_cnt] = a[a\_cnt];

      ++b\_cnt;

    }

  }

  b[b\_cnt] = '\0'; /\* Gán ký tự kết thúc chuỗi '\0' \*/

  return b;

}

int main(void){

  char a[50] = "abcdcde"; /\* Chuỗi ban đầu\*/

  puts(a);

  char b[50]; /\* Chuỗi kết quả\*/

  char chr = 'c'; /\* Ký tự cần xóa\*/

  puts(pop\_str(a,chr,b));

  return 0;

}

Kết quả:

abcdcde

abdde

### Xóa ký tự cuối

Để xóa ký tự cuối trong chuỗi C, chúng ta đơn giản dịch chuyển ký tự kết thúc chuỗi \0 tiến về phía đầu chuỗi 1 đơn vị là xong.

VD20

#include<stdio.h>

#include <string.h>// for strlen

char\* pop\_str\_last(char\* str){

    const int len = strlen(str);

    if( len == 0 ) {

        return '\0';      // Nếu chuỗi ban đầu là Null thì trả về '\0'

    } else {

        /\* Dịch chuyển ký tự kết thúc chuỗi để xóa ký tự cuối cùng\*/

        str[len-1] = '\0';

        return str;// Trả về chuỗi kết quả

    }

}

int main(void){

  char a[50] = "abcdcde";

  puts(a);

  puts(pop\_str\_last(a));

  return 0;

}

Kết quả:

abcdcde

abcdcd

### Xoá chuỗi

Ý tưởng ở đây là chúng ta tìm kiếm vị trí của chuỗi con trong chuỗi mẹ bằng hàm strstr(), sau đó tách chuỗi mẹ ra từ vị trí tìm thấy, và nối các phần này lại với nhau bằng hàm strcat().

VD21

#include <stdio.h>

#include <string.h>

char\* pop\_child\_str(char\* str1,char\* str2){

  const int len = strlen(str2);

  char str3[128] = "";

  char \*p1 = str1, \*p2;

    /\* Tạo vòng lặp để xóa hết chuỗi con \*/

  while((p2 = strstr(p1,str2)) != NULL) { /\*Tìm vị trí chuỗi con bằng hàm strstr\*/

    strncat(str3,p1,p2 - p1);   /\* Nối các phần không chứa chuỗi con \*/

    p1 = p2 + len;      /\* Dịch chuyển con trỏ sang vị trí tìm kiếm tiếp theo \*/

  }

  strcat(str3,p1);

  printf("%s\n",str3);

}

int main(void)

{

  char str1[128], str2[128];

  printf("Nhap chuoi me: ");

  scanf("%s",str1);

  printf("Nhap chuoi con can xoa: ");

  scanf("%s",str2);

  pop\_child\_str(str1,str2);

  return(0);

}

Kết quả:

Nhap chuoi me: hello\_world

Nhap chuoi con can xoa: lo

hel\_world

### Sắp xếp các ký tự của chuỗi

Bằng cách sử dụng vòng lặp for để lấy ra từng ký tự và so sánh chúng với nhau, chúng ta có thể tiến hành sắp xếp các ký tự của chuỗi trong C theo thứ tự trong bảng chữ cái.

VD22:

#include <stdio.h>

#include <string.h>

int main (void) {

  char str[256] ;

  /\* Nhập các chuỗi vào chương trình \*/

  printf("\nNhap chuoi: ");

  scanf("%s",str);

  /\* Tìm độ dài chuỗi\*/

  int length = strlen(str);

  /\* Tạo vòng lặp để tiến hành sắp xếp ký tự trong chuỗi\*/

  int i, j;

  char temp;

  for (i = 0; i < length-1; i++) {

      for (j = i+1; j < length; j++) {

         /\* So sánh lần lượt các ký tự và hoán đổi giá trị nhờ biến temp\*/

         if (str[i] > str[j]) {

            temp = str[i];

            str[i] = str[j];

            str[j] = temp;

         }

      }

  }

  printf("\nKet qua: %s \n", str);

  return 0;

}

Kết quả:

Nhap chuoi: adeszyx

Ket qua: adesxyz

**Sắp xếp các chuỗi theo thứ tự chữ cái**

Cũng bằng cách sử dụng vòng lặp for, chúng ta có thể tiến hành nhập nhiều chuỗi từ bàn phím vào chương trình và sắp xếp các chuỗi này theo theo thứ tự chữ cái bằng chương trình sau đây.

Lưu ý ở đây chúng ta so sánh các chuỗi với nhau bằng hàm strcmp(), sau đó lưu kết quả so sánh vào mảng kết quả bằng hàm strcpy() và in kết quả ra màn hình.

VD23

#include<stdio.h>

#include<string.h>

int main(void)

{

  int i, j, num;

  /\* Lệnh nhập số chuỗi cần nhập \*/

  printf("So chuoi can nhap: ");

  scanf("%d",&num);

  /\* Khai báo chuỗi sử dụng để sắp xếp \*/

  char str[num][20], tmp[20];

  /\* Nhập các chuỗi vào chương trình \*/

  printf("\nHay nhap %d chuoi vao chuong trinh\n",num);

  for (i=0;i<num;i++) {

    printf("Chuoi thu %d= ", i+1);

    scanf("%s", str[i]);

  }

  /\* Sắp xếp \*/

  for(i=1;i<num;i++){

    for(j=1;j<num;j++){

      if(strcmp(str[j-1], str[j])>0){

        strcpy(tmp, str[j-1]);

        strcpy(str[j-1], str[j]);

        strcpy(str[j], tmp);

      }

    }

  }

  /\* In kết quả sắp xếp\*/

  printf("\n");

  for (i=0;i<num;i++)

    printf("%s\n", str[i]);

  return 0;

}

Kết quả:

So chuoi can nhap: 3

Hay nhap 3 chuoi vao chuong trinh

Chuoi thu 1= akj

Chuoi thu 2= hko

Chuoi thu 3= bcd

akj

bcd

hko

### Đảo ngược ký tự trong chuỗi

Bằng cách sử dụng vòng lặp while để lấy ra từng ký tự và so sánh chúng với nhau, chúng ta có thể tiến hành đảo ngược thứ tự các từ trong chuỗi C theo thứ tự trong bảng chữ cái.

Để so sánh các ký tự với nhau, chúng ta có thể sử dụng tới toán tử so sánh, hoặc là hàm strcmp() chẳng hạn.

VD24

#include<stdio.h>

/\* Tạo hàm đảo ngược chuỗi trong C\*/

void str\_reverse(char str[]) {

  int lo = 0;

  int length;

  for (length = 0; str[length] != '\0' ; length++) {}

  while ( lo < length ) {

    --length;

    char tmp = str[lo];

    str[lo] = str[length];

    str[length] = tmp;

    ++lo;

  }

}

int main() {

    char str[100];

    /\* Nhập chuỗi cần đảo ngược từ bàn phím\*/

    printf("Nhap chuoi: ");

    scanf("%s", str);

    /\* Gọi hàm đảo ngược chuỗi\*/

    str\_reverse(str);

    /\* In kết quả\*/

    printf("Chuoi viet nguoc: %s", str);

    return 0;

}

Kết quả:

Nhap chuoi: abcxy

Chuoi viet nguoc: yxcba

### Chuyển số thành chuỗi trong C | hàm sprintf()

Tham khảo [sprintf](#_Hàm_sprintf())

VD:

#include <stdio.h>

int main(void) {

    int num1 = 88;

    float num2 = 10.8;

    //chuyển int sang string trong C

    char target1[100];

    sprintf(target1, "%d\n", num1);

    printf("%s", target1);

    //chuyển float sang string trong C

    char target2[100];

    sprintf(target2, "%f\n", num2);

    printf("%s", target2);

    return 0;

}

Kết quả:

88

10.800000

### Chuyển int sang string trong C | hàm itoa()

Lưu ý rằng itoa() không phải là một hàm chuẩn trong C, và nó có thể **không hoạt động** trong một số môi trường lập trình C.

Cú pháp của hàm itoa() để chuyển số nguyên thành chuỗi trong C như sau:

itoa (num, target, base) ;

Trong đó

* num là số nguyên cần chuyển thành chuỗi
* target là chuỗi đích để chứa num sau khi chuyển thành chuỗi
* base là cơ số chuyển đổi, nhằm xác định một giá trị số nguyên, chuyển đổi nó thành một giá trị cơ bản và lưu trữ nó trong một bộ đệm. Nếu cơ số là 10 và giá trị là âm, thì chuỗi kết quả được đặt trước dấu trừ ( -).

VD34

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(void)

{

    int number,l;

    char string[20];

    printf("Enter a number: ");

    scanf("%d", &number);

    itoa(number,string,10);

    printf("String value = %s\n", string);

    return 0;

}

Enter a number: 6512

String value = 6512

### Chuyển string sang int trong C | hàm atoi()

Hàm atoi() là một hàm có sẵn trong header file **stdlib.h**, giúp chúng ta **chuyển string sang int trong C**.

Chúng ta sử dụng hàm strcpy() để chuyển string sang int với cú pháp sau đây:

atoi(str);

Trong đó str là chuỗi cần chuyển thành số.

Nếu chuỗi chỉ định có thể chuyển sang int thì hàm atoi() sẽ chuyển chuỗi đó sang kiểu số nguyên. Tuy nhiên nếu chuỗi đó không thể chuyển sang kiểu số nguyên int thì giá trị 0 sẽ được trả về.

Lại nữa, trong trường hợp chuỗi chỉ định chứa cả thành phần **có thể và không thể** chuyển sang kiểu int thì hàm atoi() sẽ chỉ chuyển **phần string có thể chuyển thành int tính từ đầu chuỗi** mà thôi.

Và nếu như ở đầu chuỗi chỉ định chứa ký tự trắng (dấu cách, tab hay xuống dòng) thì hàm atoi() sẽ bỏ qua chúng cho tới khi tìm thấy ký tự khác ký tự trắng đầu tiên xuất hiện.

VD35

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

  /\* Chuỗi chứa số nguyên int\*/

    char str0[] = "123";

  /\* Chuỗi chứa số thực có một phần số nguyên\*/

    char str1[] = "123";

    /\* Chuỗi chứa cả phần có thể và không thể chuyển sang int\*/

    char str2[] = "-123ab";

    /\* Chuỗi chứa ký tự trắng đầu chuỗi\*/

    char str3[] = "    \n\t123ab";

    /\* Chuỗi không thể chuyển sang số nguyên\*/

    char str4[] = "a123b";

    int num0 = atoi(str0);

    int num1 = atoi(str1);

    int num2 = atoi(str2);

    int num3 = atoi(str3);

    int num4 = atoi(str4);

    printf("%s => %d\n", str0,num0);

    printf("%s => %d\n", str1,num1);

    printf("%s => %d\n", str2,num2);

    printf("%s => %d\n", str3,num3);

    printf("%s => %d\n", str4,num4);

    return 0;

}

123 => 123

123 => 123

-123ab => -123

123ab => 123

a123b => 0

### Chuyển string sang long | atol(str);

VD36

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main () {

   long val;

   char str[20];

   strcpy(str, "11111.22abac");

   val = atof(str);

   printf("%s => %ld\n", str, val);

   return(0);

}

11111.22abac => 11111

### Chuyển string sang float | atoi(str);

VD37

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main () {

   float val;

   char str[20];

   strcpy(str, "11.22abac");

   val = atof(str);

   printf("%s => %.2f\n", str, val);

   return(0);

}

11.22abac => 11.22

### Chuyển chữ hoa sang chữ thường trong C | Hàm tolower()

**Hàm tolower trong C** là một hàm có sẵn header file **ctype.h**, giúp chúng ta **chuyển chữ hoa sang chữ thường trong C**.

Cú pháp sử dụng hàm tolower() trong C như sau:

tolower(chr);

Trong đó chr là ký tự cần chuyển về chữ thường trong C.

Hàm tolower() sẽ trả về chữ thường sau khi chuyển đổi từ ký tự chỉ định nếu có thể, và trả về chính ký tự chỉ định, nếu ký tự đó không thể chuyển về chữ thường.

VD38

#include <stdio.h>

#include  <ctype.h>

int main(void){

    char c;

    /\*Trường hợp có thể chuyển về chữ thường\*/

    c = tolower('H');

    printf("%c\n", c); //h

    c = tolower('A');

    printf("%c\n", c); //a

    /\*Trường hợp không thể chuyển về chữ thường\*/

    c = tolower('a');

    printf("%c\n", c); //a

    c = tolower('8');

    printf("%c\n", c); //8

    return 0;

}

h

a

a

8

### Chuyển chữ thường sang chữ hoa trong C | Hàm toupper()

Ngược với hàm tolowertolower() thì chúng ta có hàm toupper(), với cách sử dụng tương tự nhưng chức năng thì ngược nhau.

**Hàm toupper trong C** là một hàm có sẵn header file **ctype.h**, giúp chúng ta **chữ thường sang chữ hoa trong C**.

Cú pháp sử dụng hàm toupper() trong C như sau:

toupper(chr);

Trong đó chr là ký tự cần chuyển về chữ hoa trong C.

Hàm toupper() sẽ trả về chữ hoa sau khi chuyển đổi từ ký tự chỉ định nếu có thể, và trả về chính ký tự chỉ định, nếu ký tự đó không thể chuyển về chữ hoa.

VD39

#include <stdio.h>

#include  <ctype.h>

int main(void){

    char c;

    /\*Trường hợp có thể chuyển về chữ hoa\*/

    c = toupper('h');

    printf("%c\n", c); //H

    c = toupper('a');

    printf("%c\n", c); //A

    /\*Trường hợp không thể chuyển về chữ hoa\*/

    c = toupper('H');

    printf("%c\n", c); //H

    c = toupper('!');

    printf("%c\n", c); //!

    return 0;

}

H

A

H

!

### Tự tạo hàm chuyển chữ hoa chữ thường trong C

VD40

#include <stdio.h>

/\*Hàm chuyển chữ thường thành chữ hoa trong C\*/

char upper(char c){

    if('a' <= c && c <= 'z'){

        c = c - ('a' - 'A');

    }

    return c;

}

/\*Hàm chuyển chữ hoa thành chữ thường trong C\*/

char lower(char c){

    if('A' <= c && c <= 'Z'){

        c = c + ('a' - 'A');

    }

    return c;

}

int main(void){

    char c;

    c = lower('L');

    printf("%c\n", c);

    c = upper('l');

    printf("%c\n", c);

    return 0;

}

l

L

### Kiểm tra chữ hoa chữ thường

Ý tưởng ở đây là chúng ta sẽ tiến hành kiểm tra ký tự chỉ định xem coi nó có nằm trong phạm vi các chữ hoa, hay là trong phạm vi các chữ thường không, từ đó có thể phán đoán được nó là chữ hoa hay chữ thường. Cách kiểm tra rất đơn giản, ví dụ để kiểm tra ký tự đã cho có nằm trong phạm vi các chữ thường không, chúng ta sẽ so sánh ký tự đó với hai chữ cái đầu tiên và cuối cùng trong phạm vi là a và z chẳng hạn.

VD41

/\*

 \* Hàm kiểm tra chữ hoa chữ thường trong C - Lập trình căn bản.com

 \*/

#include<stdio.h>

void upper\_or\_lower\_char(char chr){

  /\* kiểm tra xem ký tự đó là: chữ hoa, chữ thường hay ký tự khác \*/

  if((chr>= 'a') && (chr<='z')){

    printf("Chu thuong！\n");

  }

  else if((chr>= 'A') && (chr<='Z')){

    printf("Chu hoa！\n");

  }

  else {

    printf("Ky tu khac！\n");

  }

}

int main(void){

  /\* Nhập ký tự \*/

  printf("Nhap vao 1 ky tu = ");

  char chr;

  scanf("%c", &chr);

  upper\_or\_lower\_char(chr);

  return 0;

}

Nhap vao 1 ky tu = a

Chu thuong！

Nhap vao 1 ky tu = K

Chu hoa！

Nhap vao 1 ky tu = 345

Ky tu khac！

### Nhập chuỗi có khoảng trắng

Hàm scanf() trong C chỉ có tác dụng nhập các chuỗi không chứa khoảng trắng. Trong trường hợp cần nhập chuỗi chứa khoảng trắng tạo bởi dấu cách, tab hay ký tự xuống dòng, chúng ta sẽ dùng hàm fgets() để thay thế.

Hàm fgets() vốn được sử dụng để đọc và lấy dữ liệu từ file trong C, tuy nhiên chúng ta cũng có thể **nhập chuỗi có khoảng trắng trong C** bằng hàm này với cú pháp sau đây:

fgets(str, sizeof(str)+1, stdin);

Trong đó:

* str là biến để chứa chuỗi cần nhập từ bàn phím
* sizeof(str)+1 để chỉ định số ký tự lớn nhất có thể nhập bởi hàm fgets(). Ở đây hàm sizeof(str) sẽ trả về kích cỡ của chuỗi str, bao gồm cả ký tự kết thúc chuỗi \0, nên chúng ta cần cộng thêm 1 đơn vị vào.
* stdin được sử dụng như là con trỏ file trong hàm fgets(). stdin là viết tắt của file header “stdio.h”, giúp chúng ta có thể nhận đầu vào tiêu chuẩn từ bàn phím.

VD42

#include <stdio.h>

int main(void){

    printf("Nhap chuoi co ky tu trang: ");

    char str[10];

    fgets(str, sizeof(str)+1, stdin);

    printf("Chuoi vua nhap: ");

    puts(str);

    return 0;

}

Nhap chuoi co ky tu trang: hello world

Chuoi vua nhap: hello worl

### Hàm random chuỗi

Ý tưởng ở đây là, chúng ta chuẩn bị sẵn một dãy ký tự, sau đó tiến hành lấy ký tự tại một vị trí ngẫu nhiên trong dãy đó, và vị trí ngẫu nhiên này sẽ được tạo ra bởi hàm rand().

VD43

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void rand\_text(int length, char \*result) {

    int i, rand\_int;

    char char\_set[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz&quot";

    for (i = 0; i <length; i++) {

        result[i] = char\_set[rand() % sizeof(char\_set)];

    }

    result[length] = 0;

}

int main(void){

    printf("Nhap đo dai chuoi: ");

    int length;

    scanf("%d",&length);

    char result[length + 1];

    srand(time(NULL));

    rand\_text(length, result);

    printf("%s", result);

}

Nhap đo dai chuoi: 5

XtoI

Nhap đo dai chuoi: 5

v9Dbu

Chú ý ở trên, chúng ta sử dụng hàm rand() kết hợp với cả hàm srand() và time() trong dòng lệnh srand(time(NULL)); nhằm khắc phục nhược điểm của hàm rand() và tạo ra các kết quả khác nhau trong các lần thực thi chương trình.

## 3.6 Kiểm tra chắn lẻ

Chia phần trăm cho 2 lấy dư.

#include <stdio.h>

int check\_odd\_even(int n){

    //flag = 1 => số lẻ

    //flag = 0 => số chẵn

    int flag = 1;

    if( n % 2 == 0 ) flag= 0;

    return flag;

}

int main(void){

    printf("Nhap so nguyen= ");

    int n;

    scanf("%d",&n);

    int check = check\_odd\_even(n);

    if( check == 0 ) printf("%d la so chan\n", n);

    else printf("%d la so le\n", n);

    return 0;

}

Kết quả:

Nhap so nguyen= 88

88 la so chan

Nhap so nguyen= 5

5 la so le

## 3.7 Kiểm tra số nguyên

Số nguyên trong C có thể biểu diễn bởi giá trị chính xác của nó, trong khi số thực trong C lại được biểu diễn dưới dạng dấu phẩy động, điều đó có nghĩa là số thực không thể biểu diễn bởi giá trị chính xác, mà chúng ta có thể làm tròn lên hoặc làm tròn xuống số chữ cái sau phần thập phân để biểu diễn nó với các giá trị khác nhau.

Bằng cách sử dụng tính chất này, chúng ta có thể so sánh hai giá trị **làm tròn lên** và **làm tròn xuống** của số đã cho, và nếu hai giá trị này giống nhau thì số đã cho chính là một số nguyên.

Chúng ta cần sử dụng tới hai hàm trong header file **<math.h>** là hàm eil() để làm tròn lên, và hàm floor để làm tròn xuống giá trị một số trong C.

#include <stdio.h>

#include <math.h>

int check\_real\_integer\_number(float n){

    //flag = 1 => là số nguyên

    //flag = 0 => không phải là số nguyên

    int flag = 1;

    if (ceil(n) != floor(n)) flag = 0;

    return flag;

}

int main(void){

    float n;

    printf("Nhap so= ");

    scanf("%f",&n);

    int check = check\_real\_integer\_number(n);

    if( check == 1 ) printf("So vua nhap la so nguyen");

    else if( check == 0 ) printf("So vua khong phai la so nguyen");

    return 0;

}

Kết quả

Nhap so= 6

So vua nhap la so nguyen

Nhap so= 6.1

So vua nhap la so thuc

Nhap so= 0

So vua nhap la so nguyen

Liệt kê và tính tổng tất cả các số nguyên:

#include <stdio.h>

#include <math.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

int check\_real\_integer\_number(float n){

    //flag = 1 => là số nguyên

    //flag = 0 => không phải là số nguyên

    int flag = 1;

    if (ceil(n) != floor(n)) flag = 0;

    return flag;

}

int main(void){

    float array[] = {5, 4.1, 7.2, 2, 8.9, 7, 3};

    int length= SIZE\_OF\_ARRAY(array);

    int i = 0, check,total=0;

    for(i = 0; i < length; i++){

        check = check\_real\_integer\_number(array[i]);

        if( check == 1 ) {

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

            total+=array[i];

        }

    }

    printf("\nTong cac so nguyen trong mang:%d ",total);

    return 0;

}

Kết quả:

5 2 7 3

Tong cac so nguyen trong mang:17

## 3.8 Kiểm tra số nguyên tố

Số nguyên tố là số tự nhiên lớn hơn 1 không phải là tích của hai số tự nhiên nhỏ hơn. Nói cách khác, số nguyên tố là những số chỉ có đúng hai ước số là 1 và chính nó.

Ví dụ, chúng ta xem xét các số 2, 3, 6 như sau:

Số 2 chỉ có đúng hai ước số là 1 và chính nó nên là số nguyên tố

Số 3 chỉ có đúng hai ước số là 1 và chính nó nên là số nguyên tố

Số 6 có 4 ước số là 1, 2 , 3 và 6, do đó nó không phải là số nguyên tố.

VD

#include <stdio.h>

int isprime(int n){

    //flag = 0 => không phải số nguyên tố

    //flag = 1 => số nguyên tố

    int flag = 1;

    if (n <2) return flag = 0; /\*Số nhỏ hơn 2 không phải số nguyên tố => trả về 0\*/

    /\*Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không\*/

    int i = 2;

    while(i <n){

        if( n%i==0 ) {

            flag = 0;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int n;

    printf("Nhap so tu nhien= ");

    scanf("%d",&n);

    int check = isprime(n);

    if( check == 1 ) printf("%d la so nguyen to\n", n);

    else printf("%d khong phai la so nguyen to\n", n);

    return 0;

}

Kết quả:

Nhap so tu nhien= 1

1 khong phai la so nguyen to

Nhap so tu nhien= 2

2 la so nguyen to

Nhap so tu nhien= 3

3 la so nguyen to

Nhap so tu nhien= 4

4 khong phai la so nguyen to

Nhap so tu nhien= 5

5 la so nguyen to

Nhap so tu nhien= 6

6 khong phai la so nguyen to

Liệt kê tất cả các số nguyên tố nhỏ hơn n

#include <stdio.h>

int isprime(int n){

    //flag = 0 => không phải số nguyên tố

    //flag = 1 => số nguyên tố

    int flag = 1;

    if (n <2) return flag = 0; /\*Số nhỏ hơn 2 không phải số nguyên tố => trả về 0\*/

    /\*Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không\*/

    int i = 2;

    while(i <n){

        if( n%i==0 ) {

            flag = 0;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int n;

    printf(">> nhap mot so n: ");

    scanf("%d",&n);

    int i = 0, check;

    while ( i < n){

        check = isprime(i);

        if( check == 1 ) printf("%d ", i);

        ++i;

    }

    return 0;

}

Kết quả:

>> nhap mot so n: 9

2 3 5 7

## 3.9 Kiểm tra số âm, số dương trong c

int check\_negative\_positive\_number(int n){

//flag = -1 => số âm

//flag = 0 => số 0

//flag = 1 => số dương

int flag = 0;

if (n >0 ) flag = 1;

else if (n<0) flag = -1;

return flag;

}

## 3.10 Kiểm tra số chính phương

Số chính phương hay còn gọi là số hình vuông là số tự nhiên có căn bậc hai là một số tự nhiên, hay nói cách khác, số chính phương bằng bình phương của một số tự nhiên.

#include <stdio.h>

#include <math.h>

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int n;

    printf(">> nhap mot so tu nhien: ");

    scanf("%d",&n);

    int check = find\_square\_number(n);

    if( check == 1 ) printf("%d la so chinh phuong\n", n);

    else printf("%d khong phai la so chinh phuong\n", n);

    return 0;

}

Kết quả

>> nhap mot so tu nhien: 4

4 la so chinh phuong

>> nhap mot so tu nhien: 5

5 khong phai la so chinh phuong

>> nhap mot so tu nhien: 16

16 la so chinh phuong

>> nhap mot so tu nhien: 25

25 la so chinh phuong

Kiểm tra số chính phương trong C bằng căn bậc 2

Cách thứ 2 để kiểm tra số chính phương trong C là sử dụng tính chất căn bậc 2 của một số chính phương chính là một số tự nhiên. Điều đó có nghĩa, với một số bất kỳ, nếu căn bậc 2 của nó là một số tự nhiên, thì số đó chính là số chính phương.

Để tìm căn bậc 2 trong C, chúng ta cần sử dụng tới hàm sqrt() trong header file <math.h>.

#include <stdio.h>

#include <math.h>

/\*Hàm kiểm tra một số có phải là số nguyên hay không\*/

int check\_real\_integer\_number(float n){

//flag = 1 => số nguyên

//flag = 0 => số thực

int flag = 1;

if (ceil(n) != floor(n)) flag = 0;

return flag;

}

/\*Hàm kiểm tra một số có phải là số chính phương hay không\*/

int find\_square\_number(int n){

//flag = 1 => số chính phương

//flag = 0 => không phải số chính phương

int flag = 0;

//Kiểm tra căn bậc 2 của số đó có phải là số nguyên hay không

if (check\_real\_integer\_number (sqrtf(n))) flag = 1;

return flag;

}

Liệt kê tất cả các số chính phương nhỏ hơn n

#include <stdio.h>

#include <math.h>

int find\_square\_number(int n){

//flag = 1 => số chính phương

//flag = 0 => không phải số chính phương

int flag = 0;

//Tìm số bất kỳ nhỏ hơn n mà bình phương bằng n

int i = 0;

while(i <= n){

if( pow( i, 2) == n ) {

flag = 1;

break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

}

i++;

}

return flag;

}

int main(void){

int n;

printf(">> nhap mot so n: ");

scanf("%d",&n);

int i = 0, check;

while ( i < n){

check = find\_square\_number(i);

if( check == 1 ) printf("%d ", i);

++i;

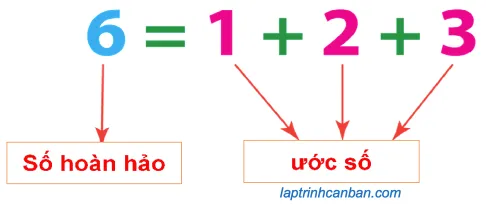
}

return 0;

}

## 3.11 Kiểm tra số hoàn hảo

Số hoàn hảo (hay còn gọi là số hoàn chỉnh, số hoàn thiện hoặc số hoàn thành) là một số nguyên dương mà tổng các ước nguyên dương chính thức của nó (số nguyên dương bị nó chia hết ngoại trừ nó) bằng chính nó.



Để kiểm tra một số có phải là số hoàn hảo hay không, chúng ta cần **tìm ra và tính tổng tất cả các ước số**, rồi so sánh xem **tổng này có bằng với số đã** cho hay không.

int find\_perfect\_num(int n){

// flag = 1 => số hoàn hảo

// flag = 0 => không phải số hoàn hảo

int flag = 0, total=0;

//Tìm tổng ước số

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

if (n % i == 0) total+=i;//Nếu là ước số thì cộng vào tổng

}

//So sánh tổng ước số với số đã cho để tìm ra số hoàn hảo

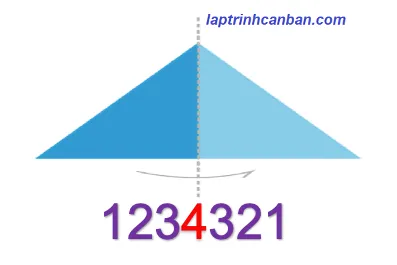
if (total == n & n!=0) flag = 1;

return flag;

}

## 3.12 Kiểm tra số đối xứng

Số nguyên n gọi là số đối xứng nếu đọc từ trái qua phải, hay từ phải qua trái đều được số giống nhau. Ví dụ: 11,121,101 là các số đối xứng.



#include <stdio.h>

/\*Hàm tìm số đảo ngược trong C\*/

int reverse\_num(int n){

  int reverse = 0;

  while (n > 0) {

    reverse = reverse \* 10 + n % 10;

    n /= 10;

  }

  return reverse;

}

/\*Hàm kiểm tra số đối xứng trong C\*/

int symmetrical\_num(int n){

    // flag = 1 => số đối xứng

    // flag = 0 => không phải số đối xứng

  int flag =0;

  if (reverse\_num(n) == n) flag = 1;

  return flag;

}

int main(void){

    int n;

    printf(">> Nhap so tu nhien= ");

    scanf("%d",&n);

    int check = symmetrical\_num(n);

    if( check == 1 ) printf("%d la so doi xung\n", n);

    else printf("%d khong phai la so doi xung\n", n);

    return 0;

}

Kết quả:

>> Nhap so tu nhien= 1234

1234 khong phai la so doi xung

>> Nhap so tu nhien= 12321

12321 la so doi xung

>> Nhap so tu nhien= 12344321

12344321 la so doi xung

## 3.13 Tìm số đảo ngược



/\*Hàm tìm số đảo ngược trong C\*/

int reverse\_num(int n){

  int reverse = 0;

  while (n > 0) {

    reverse = reverse \* 10 + n % 10;

    n /= 10;

  }

  return reverse;

}

## 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/>

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/mang-trong-c/mang-trong-c-la-gi/>

## 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

Khác với mảng 1 chiều thì **mảng 2 chiều** là **kiểu mảng chứa các mảng khác bên trong nó**. Phần tử của mảng 2 chiều không được lưu giữ trực tiếp trong mảng 2 chiều, mà được lưu giữ thông qua các mảng 1 chiều bên trong mảng 2 chiều đó. Do cấu tạo mảng như vậy nên chúng ta mới gọi các **mảng trong mảng** như thế này là mảng 2 chiều.

Mỗi phần tử trong mảng 2 chiều cần được xác định bởi một cặp index (chỉ số) là [index dọc][index ngang], trong đó **[index dọc]** để xác vị trí của mảng 1 chiều chứa nó trong mảng 2 chiều, và **[index ngang]** để xác định vị trí của nó trong mảng 1 chiều chứa nó.

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

### Lấy kích thước mảng 2 chiều

VD

#include <stdio.h>

int main(void){

    int bangdiem[][3] = {{7, 9, 8 }, {8, 6, 7}} ;

    int size = sizeof bangdiem;

    printf("%d",size);

}

//24

### Lấy độ dài của mảng 2 chiều

#include <stdio.h>

int main(void){

    int a[][3] = {{7, 9, 8 }, {8, 6, 7}} ;

     //Lấy độ dài của mảng 2 chiều

    int y = sizeof(a) / sizeof(a[0]);

    int x = sizeof(a[0]) / sizeof(a[0][0]);

    printf("Do dai mang 2 chieu: y = %d , x = %d",y, x);

}

//Do dai mang 2 chieu: y = 2 , x = 3

### Lấy số phần tử

#include <stdio.h>

int main(void){

    int bangdiem[][3] = {

        {7, 9, 8} ,

        {8, 6, 7} ,

        {5, 7, 6} ,

        {4, 9, 5} ,

        {5, 8, 7} ,

        {6, 9, 3}

    } ;

     //Lấy độ dài của mảng 2 chiều

    int y = sizeof(bangdiem) / sizeof(bangdiem[0]);

    int x = sizeof(bangdiem[0]) / sizeof(bangdiem[0][0]);

    printf("So phan tu cua mang 2 chieu: %d",y \* x);

}

//So phan tu cua mang 2 chieu: 18

## Lấy kích thước mảng trong C | toán tử sizeof()

Toán tử sizeof trả về kích thước bộ nhớ của một biến hoặc một mảng được tính bằng đơn vị byte.

#include <stdio.h>

int main(void){

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

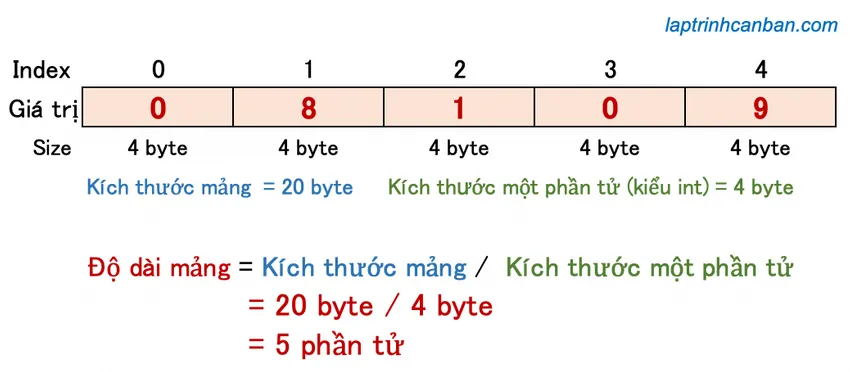
int size = sizeof a;

printf("%d",size);

}

//20

## Độ dài mảng



Cú pháp:

int length = sizeof a / sizeof(int);

Ví dụ: Lấy độ dài mảng chuỗi

#include <stdio.h>

int main(void){

char a[] = "abcdef";

int length = sizeof a / sizeof(char);

printf("%d",length);

}

//7

Ví dụ: Lấy độ dài mảng số (kiểu int)

#include <stdio.h>

int main(void){

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

int length = sizeof a / sizeof(int);

printf("%d",length);

}

//5

## Con trỏ mảng

Có 2 loại con trỏ mảng trong c là con trỏ mảng 1 chiều và con trỏ mảng đa chiều.

Con trỏ mảng biểu thị địa chỉ của điểm bắt đầu vùng lưu trữ mảng trong bộ nhớ. Địa chỉ này cũng chính là địa chỉ của phần tử đầu tiên (có index bằng 0) của mảng trong bộ nhớ.

### Khai báo con trỏ

VD: (gán địa chỉ không cần dấu &)

int nums[] = {1,2,3,4}, \*p;

p = nums;

VD: (Con trỏ biểu thị địa chỉ của điểm bắt đầu vùng lưu trữ mảng trong bộ nhớ)

#include <stdio.h>

int main(void){

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

    p = nums;

    printf("%d\n", p);

    return 0;

}

// 1378305712

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

VD

#include <stdio.h>

int main(void){

    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

    return 0;

}

array[0] address: 2039942224

array[1] address: 2039942228

array[3] address: 2039942236

### Chỉ định địa chỉ bằng con trỏ để nhập phần tử trong mảng

VD3 (scanf và printf)

#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;

}

Nhap nums[0] = 2

Nhap nums[1] = 4

Nhap nums[2] = 5

nums[0]= 2

nums[1]= 4

nums[2]= 5

### Truy xuất giá trị phần tử trong mảng bằng con trỏ

Truy cập phần tử mảng theo cách thông thường: array[index]

Truy cập phần tử mảng bằng con trỏ: \*(p + index );

Trong trường hợp index bằng 0, chúng ta có thể hiểu rằng cách viết \*p biểu thị giá trị của phần tử đầu tiên trong mảng.

Cách truy cập vào phần tử trong mảng bằng index và bằng con trỏ có thể so sánh như bảng dưới đây:

| **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) |

VD:

#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;

}

30

30

55

55

## Sao chép mảng

Trong python có thể dùng arrayA = arrayB để sao chép mảng, nhưng C thì không thể.

### Cách 1; Dùng vòng lặp for để chép từng phần tử mảng A sang mảng B .

VD4: Mảng 1 chiều

#include <stdio.h>

int main(void){

    /\*Khởi tạo chuỗi ban đầu\*/

    short data1[10] = {5,7,2,8,1,0,3,9,6,4};

    /\*Khai báo chuỗi kết quả có cùng độ dài\*/

    short data2[10];

    /\*Tạo vòng lặp, lấy và copy từng phần tử từ mảng cũ và gán vào mảng mới \*/

    for (short i = 0; i < 10; i++) {

        data2[i] = data1[i];

    }

    /\*Kiểm tra mảng mới\*/

    for(short i = 0; i < 10; i++) {

      printf("%d ", data2[i]);

    }

}

//> 5 7 2 8 1 0 3 9 6 4

VD5: Mảng 2 chiều

#include <stdio.h>

int main(void){

    /\*Khởi tạo chuỗi ban đầu\*/

    short data1[2][10] = {{5,7,2,8,1,0,3,9,6,4}, {-5,-7,-2,-8,-1,0,-3,-9,-6,-4}};

    /\*Khai báo chuỗi kết quả có cùng độ dài\*/

    short data2[2][10];

    /\*Tạo vòng lặp, lấy và copy từng phần tử từ mảng cũ và gán vào mảng mới \*/

    short i, j;

    for (i = 0; i < 2; i++){

        for (j = 0; j < 10; j++) {

            data2[i][j] = data1[i][j];

        }

    }

    /\*Kiểm tra mảng mới\*/

    for (i = 0; i < 2; i++){

        for (j = 0; j < 10; j++) {

            printf("%d ",data2[i][j]);

        }

        printf("\n");

    }

}

//  5 7 2 8 1 0 3 9 6 4

// -5 -7 -2 -8 -1 0 -3 -9 -6 -4

VD: Tạo hàm sao chép mảng

#include <stdio.h>

short array\_copy(short\* array1, short\* array2, short size)

{

    short i;

    /\*Nếu chuỗi ban đầu hoặc chuỗi kết quả là Null, kết thúc hàm và trả về 0 \*/

    if ((array1 == NULL) ||

        (array2 == NULL) ||

        (size <= 0     )   ) {

        return 0;

    }

    /\*Tạo vòng lặp, lấy và copy từng phần tử từ mảng cũ và gán vào mảng mới \*/

    for (i = 0; i < size; i++) {

        array1[i] = array2[i];

    }

    /\*Kiểm tra mảng mới\*/

    for(short i = 0; i < size; i++) {

      printf("%d ", array2[i]);

    }

    return 1;

}

int main(void){

    short data1[10] = {5,7,2,8,1,0,3,9,6,4};

    short data2[10];

    array\_copy(data2, data1, 10);

}

//> 5 7 2 8 1 0 3 9 6 4

### Cách 2: Sử dụng hàm memcpy

VD6

#include <stdio.h>

#include <string.h>

int main(void)

{

    short src[10] = {5,7,2,8,1,0,3,9,6,4};

    short dest[10];

    /\*Sao chép mảng trong C bằng hàm memcpy\*/

    memcpy((void \*)dest, (void \*)src, sizeof(src));

    /\*Kiểm tra mảng mới\*/

    for(short i = 0; i < 10; i++) {

      printf("%d ", dest[i]);

    }

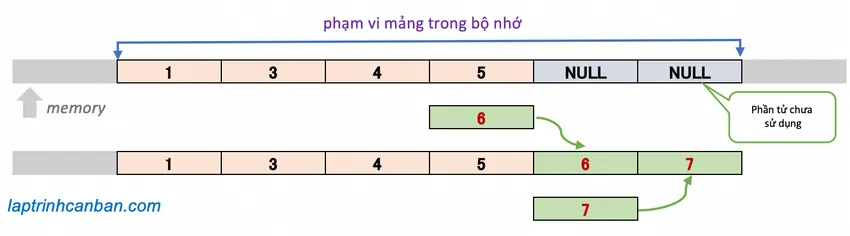
}

//> 5 7 2 8 1 0 3 9 6 4

## Thêm phần tử vào cuối mảng

Điều kiện: phần tử ở cuối mảng đó chưa được gán giá trị cụ thể. Nếu không sẽ bị tràn bộ nhớ.

Lúc này, chúng ta đơn giản chỉ cần truy cập vào các phần tử chưa được gán giá trị cụ thể, và gán giá trị mới của phần tử vào là có thể thêm phần tử vào mảng ban đầu.



VD:

#include <stdio.h>

int main(void){

    /\*Mảng ban đầu mới có 4 giá trị cụ thể được khởi tạo\*/

    /\*Các vị trí khác chưa có giá trị cụ thể và chúng ta có thể thêm phần tử vào\*/

    int arr[6] = {1, 3, 4, 5};

    printf("Mang ban dau:\n");

    for(short i = 0; i < 6; i++) printf("%d ", arr[i]);

    /\*Thêm phần tử vào các vị trí cuối mảng chưa có giá trị cụ thể\*/

    arr[4] = 4;

    arr[5] = 8;

    printf("\nMang sau khi them phan tu:\n");

    for(short i = 0; i < 6; i++) printf("%d ", arr[i]);

}

Mang ban dau:

1 3 4 5 0 0

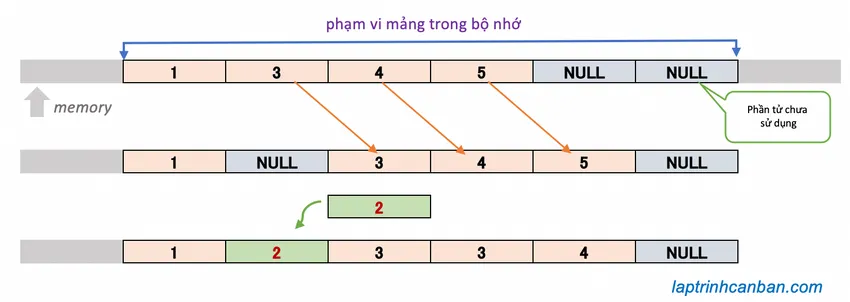
Mang sau khi them phan tu:

1 3 4 5 6 7

## Chèn phần tử vào mảng trong C vào vị trí bất kỳ

Các bước chèn phần tử vào mảng trong trường hợp này sẽ như sau:

1. Dịch chuyển các phần tử từ vị trí cần chèn về phía cuối mảng 1 đơn vị, và làm trống vị trí cần chèn
2. Chèn phần tử cần thêm vào vị trí trống mới tạo ở trên.



VD7

#include <stdio.h>

int main()

{

   int array[100], position, c, n, value;

   printf("Nhap so phan tu: ");

   scanf("%d", &n);

   printf("Nhap phan tu:\n");

   for (c = 0; c < n; c++)

      scanf("%d", &array[c]);

   printf("Mang da nhap:");

   for (c = 0; c < n; c++)

      printf("%d ", array[c]);

   printf("\nNhập vi tri chen phan tu: \n");

   scanf("%d", &position);

   printf("Nhập phan tu muon chen: \n");

   scanf("%d", &value);

   // Dịch chuyển các phần tử từ vị trí cần chèn (position) về phía sau 1 đơn vị

   // Và làm trống vị trí position

   for (c = n - 1; c >= position - 1; c--)

      array[c+1] = array[c];

   /\*Chèn phần tử vào chỗ trống vừa tạo\*/

   array[position-1] = value;

   printf("Mang sau khi chen:\n");

   for (c = 0; c <= n; c++)

      printf("%d\n", array[c]);

   return 0;

}

Nhap so phan tu: 4

Nhap phan tu:

5

6

7

9

Mang da nhap:5 6 7 9

Nhập vi tri chen phan tu:

3

Nhập phan tu muon chen:

888

Mang sau khi chen:

5

6

888

7

9

## Ghép 2 mảng

Khác với các ngôn ngữ lập trình khác thì mảng trong C có kích thước cố định được quyết định khi khai báo mảng, do đó để ghép nối 1 mảng vào mảng khác trong C, chúng ta cần **chú ý** mảng đích có đủ độ dài để chứa hết cả 2 mảng hay không. Nếu không đủ độ dài mà chúng ta vẫn cố ghép mảng, hệ quả là sự cố **tràn màn hình** có thể bị xảy ra.

Tuy nhiên cũng còn một cách khác hữu hiệu hơn, đó chính là thay vì ghép mảng 1 vào mảng 2 và lo lắng về kích thước có đủ chỗ không, thì chúng ta **chuẩn bị sẵn một mảng thứ 3, và ghép 2 mảng ban đầu vào nó**.

VD8

#include <stdio.h>

#include <string.h>

int main(void){

    int array1[] = {1,2,3};

    int array2[] = {4,5};

    /\*Tìm kích thước và chiều dài của các mảng cần ghép\*/

     int size1 = sizeof array1,  length1 = sizeof array1/ sizeof(int);

     int size2 = sizeof array2,  length2 = sizeof array2/ sizeof(int);

    /\* Khai báo mảng kết quả với độ dài đủ để chứa 2 mảng trên\*/

    int result[length1 + length2];

     /\* Copy mảng đầu tiên vào mảng kết quả\*/

    memcpy(result,array1,size1);

    /\* Copy mảng thứ 2 vào vị trí cuối mảng 1 trong mảng kết quả\*/

    memcpy(&result[length1],array2,size2);

    /\*Kiểm tra kết quả ghép nối mảng\*/

    for(short i = 0; i < 5; i++) {

      printf("%d ", result[i]);

    }

}

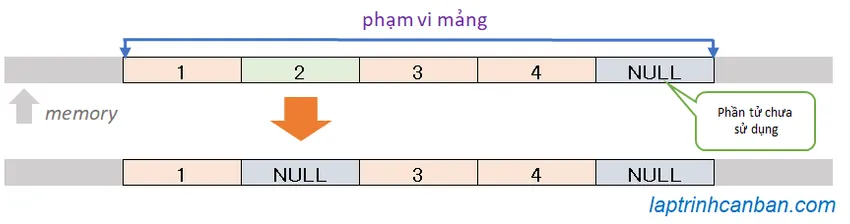
//> 1 2 3 4 5

## Xóa 1 phần tử trong mảng

Khác với các ngôn ngữ lập trình khác thì mảng trong C có kích thước cố định được quyết định khi khai báo mảng, và chúng ta không thể thay đổi kích thước của mảng sau khi khai báo.

Bởi vậy, chúng ta không thể xóa phần tử trong mảng C, sau khi đã tạo nó. Điều duy nhất chúng ta có thể làm, đó là **thay thế giá trị các phần tử cần xóa thành giá trị NULL**, và ghi nhớ các phần tử này chưa được sử dụng trong chương trình mà thôi.

Lưu ý giá trị NULL trong C nếu ở dạng char thì là ký tự \0, và nếu ở dạng int thì là **số 0**. Tuy nhiên về bản chất thì NULL có nghĩa là **ký tự rỗng** có nghĩa là **không tồn tại giá trị**.



/\*mảng đã sử dụng hết phần tử\*/

int int\_arr2[4] = {1,2,3,4}; // 1,2,3,4

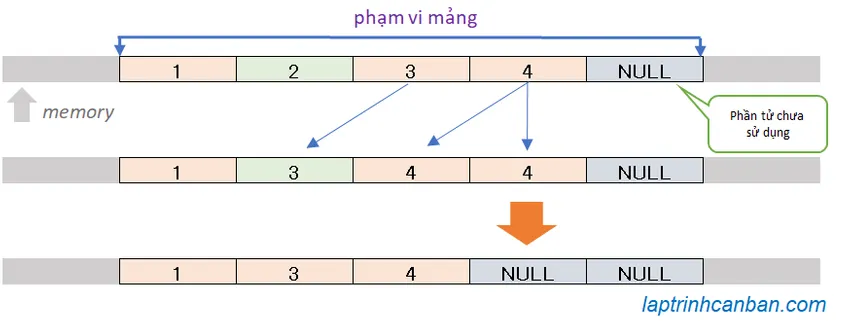
int\_arr2[2] = NULL; // 1,2,0,4

/\*mảng có phần tử chưa sử dụng\*/

int int\_arr[6] = {1,2,3,4}; // 1,2,3,4,0,0

int\_arr[2] = NULL; // {1,2,0,4,0,0}

Tuy nhiên sau khi thay thế phần tử cần xóa bằng giá trị NULL bằng cách ở trên, thì phần tử này sẽ vẫn nằm ở giữa các phần tử khác trong mảng, gây nên bất tiện khi sử dụng mảng. Do vậy chúng ta sẽ cần phải chuyển phần tử này về cuối mảng để dễ sử dụng hơn bằng cách thứ 2 sau đây:



Các bước chúng ta thực hiện sẽ là:

1. Thay đổi giá trị của tất cả các phần tử từ vị trí cần xóa trong mảng, bằng giá trị của phần tử đứng đằng sau nó.
2. Thay đổi giá trị của phần tử cuối cùng thành NULL.

Sau khi tiến hành xóa 1 phần tử trong mảng như trên, mảng ban đầu sẽ bao gồm các phần tử còn lại ở đầu mảng, và các phần tử NULL ở cuối mảng.

Nếu bạn muốn **lấy các phần tử không bị xóa** từ mảng ban đầu, chúng ta đơn giản tạo một mảng mới, sau đó sử dụng vòng lặp để lấy các phần tử chưa bị xoá từ đầu mảng ban đầu là xong.

Và chúng ta viết hàm để thực hiện việc **xóa 1 phần tử trong mảng C** như sau:

VD9

#include <stdio.h>

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm xóa 1 phần tử trong mảng\*/

int delete\_one\_element\_from\_array(int array[], int length, int position){

   int count = 0;

    /\* Dịch phần tử về đầu mảng từ vị trí xóa \*/

    for(int i = position; i < length-1; i++) {

        array[i] = array[i+1];

    }

    /\*Thay đổi giá trị phần tử cuối cùng thành NULL\*/

    array[length-1] = 0;

    count++;

    return count;

}

int main(void){

   int array[100], position, length;

   printf("Nhap so phan tu: ");

   scanf("%d", &length);

   printf("Nhap phan tu:\n");

   for (int i = 0; i < length; i++)

      scanf("%d", &array[i]);

   printf("Nhap vi tri xoa: ");

   scanf("%d", &position);

    printf("Mang truoc khi xoa:\n");

    show\_array(array,length);

    int count= delete\_one\_element\_from\_array(array,length, position);

    printf("Mang sau khi xoa:\n");

    show\_array(array,length);

    printf("So phan tu bi xoa: %d",count);

    /\*Tạo một mảng mới để lưu kết quả xóa phần tử từ mảng ban đầu\*/

    int n = length -count;

    int result[n];

    for(short i = 0; i < n; i++)  result[i] = array[i];

    printf("\nMang ket qua:\n");

    show\_array(result,n);

    return 0;

}

Nhap so phan tu: 5

Nhap phan tu:

0

1

2

3

4

Nhap vi tri xoa: 3

Mang truoc khi xoa:

0 1 2 3 4

Mang sau khi xoa:

0 1 2 4 0

So phan tu bi xoa: 1

Mang ket qua:

0 1 2 4

## Xoá nhiều phần tử

VD10

#include <stdio.h>

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm xóa 1 phần tử trong mảng\*/

int delete\_one\_element\_from\_array(int array[], int length, int position){

   int count = 0;

    /\* Dịch phần tử về đầu mảng từ vị trí xóa \*/

    for(int i = position; i < length-1; i++) {

        array[i] = array[i+1];

    }

    /\*Thay đổi giá trị phần tử cuối cùng thành NULL\*/

    array[length-1] = 0;

    count++;

    return count;

}

/\*Tạo hàm xóa nhiều phần tử trong mảng\*/

int delete\_elements\_from\_array(int array[], int length, int positions[], int delete\_num){

    int n = 0;

    int count=0;

    for (int i=0; i< delete\_num; i++){

        int position= positions[i] - count;

        n += delete\_one\_element\_from\_array(array,length, position);

        count++;

    }

    return n;

}

int main(void){

   int array[100], positions[100], length, delete\_num;

   printf("Nhap so phan tu: ");

   scanf("%d", &length);

   printf("Nhap phan tu:\n");

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

   printf("Nhap so phan tu can xoa: ");

   scanf("%d", &delete\_num);

   printf("Nhap vi tri xoa:\n");

   for (int i = 0; i < delete\_num; i++) scanf("%d", &positions[i]);

    printf("Mang truoc khi xoa:\n");

    show\_array(array,length);

    int count= delete\_elements\_from\_array(array,length, positions,delete\_num);

    printf("Mang sau khi xoa:\n");

    show\_array(array,length);

    printf("So phan tu bi xoa: %d\n",count);

    /\*Tạo một mảng mới để lưu kết quả xóa phần tử từ mảng ban đầu\*/

    int n = length -count;

    int result[n];

    for(short i = 0; i < n; i++)  result[i] = array[i];

    printf("Mang ket qua:\n");

    show\_array(result,n);

    return 0;

}

## Xóa toàn bộ mảng

Thông thường khi tạo mảng trong C, chúng ta sẽ dùng tới cách khai báo hoặc khởi tạo như sau:

int num[4];

char chars[] = "love";

Rất tiếc là với cách tạo mảng như trên, do sau khi tạo mảng này thì một vùng có kích thước cố định với kích thước mảng đã được tạo ra trong bộ nhớ máy tính để lưu mảng, và trong trường hợp này thì chúng ta không thể xóa mảng đã tạo được.

Trường hợp duy nhất mà chúng ta có thể **xóa mảng trong C** đó là khi chúng ta đã **cấp phát bộ nhớ động** để lưu mảng đã tạo ra, như các cách sau:

int \*A=malloc(20 \* sizeof(int));

//or

int \*A=calloc(20, sizeof(int));

Nếu bạn đã tạo mảng trong C bằng cách *cấp phát bộ nhớ động* cho nó, chúng ta có thể dùng hàm free() để giải phóng bộ nhớ động này, và qua đó có thể **xóa mảng trong C** sau khi tạo nó.

int \*A=malloc(20 \* sizeof(int));

free(A);

## Tách mảng âm dương

Để tách các số âm/dương ra 1 mảng khác từ mảng cho trước, chúng ta đơn giản chuẩn bị sẵn 2 mảng chứa kết quả phép tách, một mảng để chứa các số âm và một mảng để chứa các số dương được tách ra từ mảng ban đầu.

VD11

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

int main(void){

    //Khai báo 2 mảng chứa kết quả

    int negative\_array[100], positive\_array[100];

    //Khởi tạo mảng cần tách ra các số âm dương

    int array[] = {-5, 4, 0 , 1, -2 ,9 , 7,-8};

    //Sử dụng macro ở trên để lấy độ dài mảng

    int length = SIZE\_OF\_ARRAY(array);

    int negative\_count = 0, positive\_count = 0;

    for (int i = 0; i < length; i++){

        //Kiểm tra phần tử là số âm hay dương

        //Và gán số tìm thấy vào các mảng âm dương tương ứng

        //Cũng như tăng dần số phần tử trong mảng kết quả

        if (array[i] < 0) {

            negative\_array[negative\_count]=array[i];

            ++ negative\_count;

        } else if (array[i] > 0) {

            positive\_array[positive\_count]=array[i];

            ++ positive\_count;

        }

    }

    //In ra mảng âm dương đã tách ra

    show\_array(negative\_array, negative\_count);

    show\_array(positive\_array, positive\_count);

    return 0;

}

-5 -2 -8

4 1 9 7

## Tách mảng chẵn lẻ

VD

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

    printf("\n");

}

/\*Tạo hàm kiểm tra số chẵn lẻ trong C\*/

int check\_odd\_even(int n){

    //flag = 1 => số lẻ

    //flag = 0 => số chẵn

    int flag = 1;

    if( n % 2 == 0 ) flag= 0;

    return flag;

}

int main(void){

    //Khai báo 2 mảng chứa kết quả

    int odd\_array[100], even\_array[100];

    //Khởi tạo mảng cần tách ra các số chẵn lẻ

    int array[] = {-2,-1,0,1,2,3,4,5,6,7,8};

    //Sử dụng macro ở trên để lấy độ dài mảng

    int length = SIZE\_OF\_ARRAY(array);

    int odd\_count = 0, even\_count = 0;

    for (int i = 0; i < length; i++){

        //Kiểm tra phần tử là số chẵn hay lẻ

        //Và gán số tìm thấy vào các mảng chẵn lẻ tương ứng

        //Cũng như tăng dần số phần tử trong mảng kết quả

        if (check\_odd\_even(array[i]) == 1) {

          odd\_array[odd\_count]=array[i];

          ++ odd\_count;

        } else {

          even\_array[even\_count]=array[i];

          ++ even\_count;

        }

    }

    //In ra mảng chẵn lẻ đã tách ra

    show\_array(odd\_array, odd\_count);

    show\_array(even\_array, even\_count);

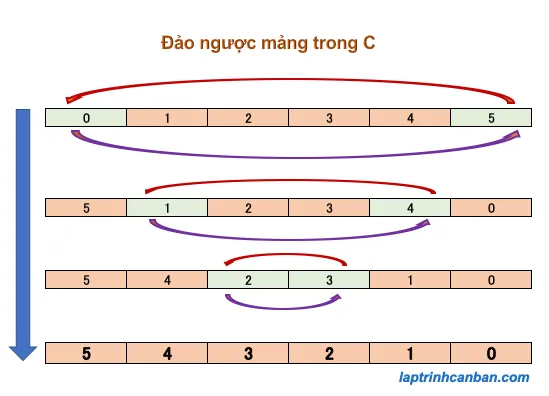
    return 0;

}

-1 1 3 5 7

-2 0 2 4 6 8

## Đảo ngược dãy số



VD12:

#include <stdio.h>

//Định nghĩa maco có tác dụng hoán đổi 2 phần tử

#define SWAP(type,x,y) do{type tmp = x; x = y; y = tmp;}while(0)

//Tạo hàm đảo ngược mảng

void reverse(int\* array, int size)

{

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

        SWAP(int, array[i], array[size - i - 1]);

    }

}

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

int main(void)

{

    int array1[] = {11, 22, 33, 44,};

    int array2[] = {0, 1, 2, 3, 4, 5};

    //Lấy độ dài mảng ban đầu

    int size1 = sizeof array1 / sizeof(int);

    int size2 = sizeof array2 / sizeof(int);

    reverse(array1, size1);

    reverse(array2, size2);

    //in mảng kết quả

    show\_array(array1,size1);

    show\_array(array2,size2);

    return 0;

}

44 33 22 11

5 4 3 2 1 0

## Sắp xếp mảng tăng dần trong C bằng hàm qsort

**Hàm qsort trong C** là một hàm có sẵn trong header file **stdlib.h**, giúp chúng ta **sắp xếp mảng trong C**. Cú pháp sử dụng hàm qsort() trong C như sau:

qsort(array, length, size, compare\_fuct);

Trong đó:

* array là mảng cần so sánh
* length là độ dài (số phần tử) của mảng đó. Xem thêm: [Độ dài mảng trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/mang-trong-c/do-dai-mang-trong-c).
* size là kích thước một phần tử trong mảng. Ví dụ như kiểu int là 4 byte, char là 1 byte chẳng hạn. Xem thêm: [Kiểu dữ liệu trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/bien-trong-c/kieu-du-lieu-trong-c/).
* compare\_fuct là hàm so sánh để quyết định thứ tự sắp xếp

Để **quyết định thứ tự sắp xếp mảng** bằng hàm qsort(), chúng ta cần sử dụng tới một hàm trung gian là **hàm so sánh** compare\_fuct giúp tìm ra **quan hệ lớn nhỏ giữa hai phần tử**.

Trong trường hợp sắp xếp mảng tăng dần trong C thì chúng ta sẽ định nghĩa hàm so sánh này như sau

int compareIntAsc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

VD13

#include <stdio.h>

#include <stdlib.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

int main(void){

    int array1[] = {5, 4, 7, 2, 8, 7, 3};

    int array2[] = {99, 4, 5, 2, 80, 7, 3};

    /\*Sử dụng hàm qsort để sắp xếp mảng tăng dần\*/

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntAsc);

    /\*Xem kết quả sắp xếp mảng\*/

    show\_array(array1, SIZE\_OF\_ARRAY(array1));

    show\_array(array2, SIZE\_OF\_ARRAY(array2));

    return 0;

}

2 3 4 5 7 7 8

2 3 4 5 7 80 99

## Sắp xếp mảng giảm dần trong C bằng hàm qsort

#include <stdio.h>

#include <stdlib.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh giảm dần sử dụng trong hàm qsort\*/

int compareIntDesc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return bNum - aNum;

}

int main(void){

    int array1[] = {5, 4, 7, 2, 8, 7, 3};

    int array2[] = {99, 4, 5, 2, 80, 7, 3};

    /\*Sử dụng hàm qsort để sắp xếp mảng giảm dần\*/

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntDesc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntDesc);

    /\*Xem kết quả sắp xếp mảng\*/

    show\_array(array1, SIZE\_OF\_ARRAY(array1));

    show\_array(array2, SIZE\_OF\_ARRAY(array2));

    return 0;

}

8 7 7 5 4 3 2

99 80 7 5 4 3 2

## Tìm min, max trong mảng

#include <stdio.h>

#include <assert.h>

/\*Hàm tìm max trong mảng C\*/

int maxElement(const int\* array, size\_t size){

    /\*Kiểm tra và báo lỗi nếu đối số truyền vào không thoả mãn điều kiện chạy hàm\*/

    assert(array != NULL);

    assert(size >= 1);

    //Giả định giá trị lớn nhất là giá trị đầu tiên của mảng.

    int max = array[0];

    /\*So sánh từng phần tử trong mảng với giá trị đầu tiên để tìm ra giá trị lớn nhất\*/

    for (size\_t i = 1; i < size; ++i) {

        if (max < array[i]) { //Thay đổi giá trị max nếu tìm ra số lớn hơn

            max = array[i];

        }

    }

    return max;

}

/\*Hàm tìm min trong mảng C\*/

int minElement(const int\* array, size\_t size){

    /\*Kiểm tra và báo lỗi nếu đối số truyền vào không thoả mãn điều kiện chạy hàm\*/

    assert(array != NULL);

    assert(size >= 1);

    //Giả định giá trị nhỏ nhất là giá trị đầu tiên của mảng.

    int min = array[0];

    /\*So sánh từng phần tử trong mảng với giá trị đầu tiên để tìm ra giá trị nhỏ nhất\*/

    for (size\_t i = 1; i < size; ++i) {

        if (min > array[i]) { //Thay đổi giá trị min nếu tìm ra số nhỏ hơn

            min = array[i];

        }

    }

    return min;

}

int main(void){

    int array[100], n;

    printf(">>Nhap so phan tu: ");

    scanf("%d", &n);

    printf(">>Nhap phan tu:\n");

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

    /\*Tìm max trong mảng\*/

    int max = maxElement(array,n);

    printf("max= %d\n", max);

    /\*Tìm min trong mảng\*/

    int min = minElement(array,n);

    printf("min= %d\n", min);

}

>>Nhap so phan tu: 5

>>Nhap phan tu:

2

5

8

1

6

max= 8

min= 1

## So sánh mảng trong C bằng cách đối chiếu từng cặp phần tử

VD15

#include <assert.h>

#include <stdbool.h>

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Hàm so sánh 2 mảng trong C\*/

int array\_equal(const int\* array1, size\_t size1, const int\* array2, size\_t size2){

    //flag = 1 => 2 mảng đã cho giống nhau

    //flag = 0 => 2 mảng đã cho khác nhau

    int flag = 1;

    /\*Kiểm tra đối số truyền vào mảng có thỏa mãn điều kiện chạy hàm không\*/

    assert(array1 != NULL);

    assert(array2 != NULL);

    assert(size1 != 0);

    assert(size2 != 0);

    /\*Nếu số phần tử của 2 mảng khác nhau, thì chúng sẽ khác nhau\*/

    if (size1 != size2)  return flag = 0;

    /\*Kiểm tra từng cặp phần tử tương ứng xem có cặp nào khác nhau không\*/

    for (size\_t i = 0; i < size1; ++i) {

        if (array1[i] != array2[i])  return flag = 0;

    }

    return flag;

}

int main(void)

{

    int array1[] = {0, 1, 2, 3, 4};

    int array2[] = {0, 1, 4};

    int array3[] = {0, 1, 2, 3, 4};

    int array4[] = {0, 1, 2, 3, 4, 5};

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array1, SIZE\_OF\_ARRAY(array1)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array2, SIZE\_OF\_ARRAY(array2)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array3, SIZE\_OF\_ARRAY(array3)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array4, SIZE\_OF\_ARRAY(array4)));

    return 0;

}

1

0

1

0

## So sánh mảng trong C bằng hàm memcmp

**Hàm memcmp** là một hàm trong header file **string.h**, vốn được sử dụng để so sánh nội dung của các vùng trên bộ nhớ với nhau. Ứng dụng hàm này, chúng ta có thể so sánh các vùng trên bộ nhớ được sử dụng để chứa các mảng, và qua đó có thể so sánh gián tiếp mảng trong C.

Chúng ta sử dụng hàm memcmp với cú pháp sau đây:

int memcmp(const void\* s1, const void\* s2, size\_t size);

Trong đó:

* s1, s2 là tên hai mảng cần so sánh, và chúng ta không chỉ định trực tiếp 2 mảng này trong đối số của hàm, mà sẽ sử dụng con trỏ mảng của chúng.
* size là phạm vi so sánh trên bộ nhớ (byte). Lưu ý đây là phạm vi tính bởi byte chứ không phải là số phần tử trong mảng đâu nhé.

Hàm memcmp thường được sử dụng để so sánh quan hệ lớn nhỏ giữa 2 vùng trong bộ nhớ, tuy nhiên bằng cách kiểm tra giá trị trả về của nó là 0 hay là một số khác 0 thì chúng ta cũng có thể so sánh xem 2 mảng trong C có giống nhau hay không.

VD16

#include <assert.h>

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

bool array\_equal(const int\* array1, size\_t size1, const int\* array2, size\_t size2)

{

    assert(array1 != NULL);

    assert(array2 != NULL);

    assert(size1 != 0);

    assert(size2 != 0);

    /\*Nếu số phần tử của 2 mảng khác nhau, thì chúng sẽ khác nhau\*/

    if (size1 != size2) {

        return false;

    }

    /\*So sánh 2 vùng bộ nhớ chứa 2 mảng trên bằng hàm memcmp\*/

    return memcmp(array1, array2, size1 \* sizeof(int)) == 0;

}

int main(void)

{

    int array1[] = {0, 1, 2, 3, 4};

    int array2[] = {0, 1, 4};

    int array3[] = {0, 1, 2, 3, 4};

    int array4[] = {0, 1, 2, 3, 4, 5};

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array1, SIZE\_OF\_ARRAY(array1)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array2, SIZE\_OF\_ARRAY(array2)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array3, SIZE\_OF\_ARRAY(array3)));

    printf("%d\n", array\_equal(array1, SIZE\_OF\_ARRAY(array1), array4, SIZE\_OF\_ARRAY(array4)));

    return 0;

}

1

0

1

0

## Kiểm tra và lấy phần tử trùng trong mảng

### Kiểm tra phần tử trùng trong mảng C bằng cách so sánh từng phần tử

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra phần tử trùng trong mảng C\*/

int array\_is\_unique(const int\* array, size\_t size){

    //flag =  1 =>  tồn tại phần tử trùng nhau

    //flag =  0 =>  không tồn tại phần tử trùng nhau

    int flag = 0;

    for (size\_t i = 0; i < size - 1; ++i) {

        for (size\_t j = i + 1; j < size; ++j) {

            if (array[i] == array[j]) {

                flag = 1;

                break;

            }

        }

    }

    return flag;

}

int main(void){

    //Khởi tạo mảng cần kiểm tra phần tử trùng nhau

    int array1[] = {1,2,3,3,2,5}; //Mảng chứa phần tử trùng nhau

    int array2[] = {1,2,3,4,5,6,7}; //Mảng không chứa phần tử trùng nhau

    //Kiểm tra phần tử trùng nhau

    int check1 = array\_is\_unique(array1,SIZE\_OF\_ARRAY(array1));

    if (check1 == 1) printf("ton tai phan tu trung nhau");

    else printf("khong ton tai phan tu trung nhau");

    printf("\n");

    //Kiểm tra phần tử trùng nhau

    int check2 = array\_is\_unique(array2,SIZE\_OF\_ARRAY(array2));

    if (check2 == 1) printf("ton tai phan tu trung nhau");

    else printf("khong ton tai phan tu trung nhau");

    return 0;

}

ton tai phan tu trung nhau

khong ton tai phan tu trung nhau

### Kiểm tra phần tử trùng trong mảng C bằng cách sắp xếp mảng

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm kiểm tra phần tử trùng trong mảng C\*/

int array\_is\_unique(const int\* array, size\_t size){

    //flag =  1 =>  tồn tại phần tử trùng nhau

    //flag =  0 =>  không tồn tại phần tử trùng nhau

    int flag = 0;

    for (size\_t i = 0; i < size - 1; ++i) {

        if (array[i] == array[i + 1]) {

            flag = 1;

            break;

        }

    }

    return flag;

}

int main(void){

    //Khởi tạo mảng cần kiểm tra phần tử trùng nhau

    int array1[] = {1,2,3,3,2,5}; //Mảng chứa phần tử trùng nhau

    int array2[] = {1,2,3,4,5,6,7}; //Mảng không chứa phần tử trùng nhau

    //Sắp xếp mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntAsc);

    //Kiểm tra phần tử trùng nhau

    int check1 = array\_is\_unique(array1,SIZE\_OF\_ARRAY(array1));

    if (check1 == 1) printf("ton tai phan tu trung nhau");

    else printf("khong ton tai phan tu trung nhau");

    printf("\n");

    int check2 = array\_is\_unique(array2,SIZE\_OF\_ARRAY(array2));

    if (check2 == 1) printf("ton tai phan tu trung nhau");

    else printf("khong ton tai phan tu trung nhau");

    return 0;

}

ton tai phan tu trung nhau

khong ton tai phan tu trung nhau

### Lấy phần tử trùng trong mảng

VD17

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Thay đổi hàm kiểm tra phần tử trùng nhau ở phương pháp một\*/

/\*Và tạo hàm lấy phần tử trùng trong mảng C\*/

void take\_duplicate\_element(const int\* array, size\_t size){

    int result[100], count=0;

    for (size\_t i = 0; i < size - 1; ++i) {

        for (size\_t j = i + 1; j < size; ++j) {

            if (array[i] == array[j]) {

            result[count]=array[i];

              ++ count;

            }

        }

    }

    show\_array(result, count);

}

int main(void){

    int array1[] = {1,2,3,3,2,5};

    int array2[] = {1,1,2,3,2,5,5,8,9,6};

    //Lấy phần tử trùng nhau

   take\_duplicate\_element(array1,SIZE\_OF\_ARRAY(array1));

   take\_duplicate\_element(array2,SIZE\_OF\_ARRAY(array2));

    return 0;

}

2 3

1 2 5

### Xóa phần tử trùng trong mảng C khi mảng đã được sắp xếp

VD18

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm xoá phần tử trùng trong mảng C\*/

size\_t array\_unique(int\* array, size\_t size)

{

    size\_t end = 0;

    for (size\_t i = 1; i < size; ++i) {

        if (array[i] != array[end]) { //Tìm thấy phần tử trùng nhau

            ++end; //Tăng dần vị trí đầu mảng để gán phần tử trùng nhau

            array[end] = array[i]; //Ghi đè phần tử trùng nhau vào vị trí đầu mảng

        }

    }

    return end + 1;

}

int main(void)

{

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    int array2[] = { 7, 2, 6, 7, 7 };

    int array3[] = { 7, 7, 7 };

    // Sắp xếp các mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntAsc);

    qsort(array3, SIZE\_OF\_ARRAY(array3), sizeof(int), compareIntAsc);

    //Xoá phần tử trùng nhau trong các mảng

    size\_t size1 = array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    size\_t size2 = array\_unique(array2, SIZE\_OF\_ARRAY(array2));

    size\_t size3 = array\_unique(array3, SIZE\_OF\_ARRAY(array3));

    //in mảng sau khi đã xoá phần tử trùng nhau

    show\_array(array1, size1);

    show\_array(array2, size2);

    show\_array(array3, size3);

    return 0;

}

2 4 6 7 8 9

2 6 7

7

### Xóa phần tử trùng trong mảng C khi mảng chưa được sắp xếp

VD19

#include <stdio.h>

#include <string.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm xoá phần tử trùng trong mảng C\*/

size\_t array\_unique(int\* array, size\_t size)

{

    for (size\_t i = 0; i < size - 1; ++i) {

        for (size\_t j = i + 1; j < size; ++j) {

            if (array[i] == array[j]) { //Tìm thấy phần tử trùng nhau

                // Ghi đè array[j] bằng phạm vi từ array[j + 1] đến cuối mảng, qua đó xoá array[j]

                memmove(&array[j], &array[j + 1], sizeof(int) \* (size - j - 1));

                --size; // Do đã xoá array[j] nên trừ độ dài mảng đi một đơn vị

                // Do có khả năng 3 phần tử trùng nhau xuất hiện liên tiếp

                // Nên chúng ta cần kiểm tra lại vị trí j bằng cách giảm giá trị của j 1 đơn vị

                --j;

            }

        }

    }

    return size;

}

int main(void)

{

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    int array2[] = { 7, 2, 6, 7, 7 };

    int array3[] = { 7, 7, 7 };

    //Xoá phần tử trùng nhau trong các mảng

    size\_t size1 = array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    size\_t size2 = array\_unique(array2, SIZE\_OF\_ARRAY(array2));

    size\_t size3 = array\_unique(array3, SIZE\_OF\_ARRAY(array3));

    //In các phần tử không trùng nhau đã được chuyển lên đầu mảng

    show\_array(array1, size1);

    show\_array(array2, size2);

    show\_array(array3, size3);

    return 0;

}

7 2 6 4 9 8

7 2 6

7

### Đếm số phần tử giống nhau trong mảng

VD20

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm đếm số phần tử giống nhau trong mảng C\*/

int take\_duplicate\_element(const int\* array, size\_t size){

    int result[100], count=0;

    for (size\_t i = 0; i < size - 1; ++i) {

        for (size\_t j = i + 1; j < size; ++j) {

            if (array[i] == array[j]) { //Tìm thấy phần tử giống nhau

            result[count]=array[i];

            ++ count;

            }

        }

    }

    show\_array(result,count);

    return count;

}

int main(void){

    int array1[] = {1,2,3,3,2,5};

    int array2[] = {1,1,2,3,2,5,5,8,9,6};

    //Lấy phần tử trùng nhau

   int check1 = take\_duplicate\_element(array1,SIZE\_OF\_ARRAY(array1));

   printf("So phan tu giong nhau: %d\n",check1);

   int check2 = take\_duplicate\_element(array2,SIZE\_OF\_ARRAY(array2));

   printf("So phan tu giong nhau: %d\n",check2);

    return 0;

}

2 3

So phan tu giong nhau: 2

1 2 5

So phan tu giong nhau: 3

### Đếm số phần tử khác nhau trong mảng

Để đếm số phần tử khác nhau trong mảng C, chúng ta đơn giản xóa đi tất cả các phần tử trùng trong mảng, rồi đếm số phần tử khác nhau còn lại trong mảng là xong.

VD21

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm xoá phần tử trùng trong mảng C\*/

size\_t array\_unique(int\* array, size\_t size)

{

    size\_t end = 0;

    for (size\_t i = 1; i < size; ++i) {

        if (array[i] != array[end]) { //Tìm thấy phần tử trùng nhau

            ++end; //Tăng dần vị trí đầu mảng để gán phần tử trùng nhau

            array[end] = array[i]; //Ghi đè phần tử trùng nhau vào vị trí đầu mảng

        }

    }

    show\_array(array, end+1);

    return end + 1;

}

int main(void)

{

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    int array2[] = { 7, 2, 6, 7, 7 };

    int array3[] = { 7, 7, 7 };

    // Sắp xếp các mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntAsc);

    qsort(array3, SIZE\_OF\_ARRAY(array3), sizeof(int), compareIntAsc);

    //Xoá phần tử trùng nhau và đếm số phần tử khác nhau trong kết quả

    size\_t size1 = array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    printf("So phan tu khac nhau: %d\n", size1);

    size\_t size2 = array\_unique(array2, SIZE\_OF\_ARRAY(array2));

    printf("So phan tu khac nhau: %d\n", size2);

    size\_t size3 = array\_unique(array3, SIZE\_OF\_ARRAY(array3));

    printf("So phan tu khac nhau: %d\n", size3);

    return 0;

}

2 4 6 7 8 9

So phan tu khac nhau: 6

2 6 7

So phan tu khac nhau: 3

7

So phan tu khac nhau: 1

### Đếm số lần xuất hiện của một phần tử trong mảng

Để đếm số lần xuất hiện của một phần tử trong mảng C, ví dụ như là đếm số lần xuất hiện của phần tử x trong mảng có n phần tử chẳng hạn, chúng ta đơn giản tạo một vòng lặp và so sánh phần tử đó với từng phần tử trong mảng xem có bao nhiêu phần tử trong mảng giống nó là xong.

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm đếm số lần xuất hiện của một phần tử trong mảng C\*/

int count\_element\_in\_array(int\* array, int size, int x){

    int count = 0;

    for(int i=0;i<size;i++){

      if(array[i]==x) //Tìm thấy phần tử giống x trong mảng thì cộng biến đếm

        count ++;

    }

    return count;

}

int main(void){

    int array[] = { 7, 2, 6, 7, 4, 9, 8 };

    int x = 7;

    int count = count\_element\_in\_array(array, SIZE\_OF\_ARRAY(array) , x);

    printf("Phan tu %d xuat hien %d lan\n",x,count);

    x = 4;

    count = count\_element\_in\_array(array, SIZE\_OF\_ARRAY(array) , x);

    printf("Phan tu %d xuat hien %d lan",x,count);

    return 0;

}

Phan tu 7 xuat hien 2 lan

Phan tu 4 xuat hien 1 lan

### Đếm số lần xuất hiện của các phần tử trong mảng

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm đếm số lần xuất hiện của các phần tử trong mảng C\*/

void array\_unique(int\* array, size\_t size){

    int count =1;

    for (size\_t i = size-1; i > 0; --i) {

        //printf("Phan tu %d",array[i-1]);

        if (array[i] == array[i-1]) ++count; //Tìm thấy phần tử trùng nhau

        else{

             printf("Phan tu %d xuat hien %d lan\n",array[i], count);

             count = 1;

        }

    }

    printf("Phan tu %d xuat hiẹn %d lan\n",array[0], count);

}

int main(void)

{

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    // Sắp xếp các mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    show\_array(array1, SIZE\_OF\_ARRAY(array1));

    //Đếm số lần xuất hiện của các phần tử trong mảng C

    array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    return 0;

}

2 4 6 7 7 8 9

Phan tu 9 xuat hien 1 lan

Phan tu 8 xuat hien 1 lan

Phan tu 7 xuat hien 2 lan

Phan tu 6 xuat hien 1 lan

Phan tu 4 xuat hien 1 lan

Phan tu 2 xuat hiẹn 1 lan

## Liệt kê phần tử

### Liệt kê các phần tử duy nhất trong mảng

VD24

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm xoá phần tử trùng lặp trong mảng đã sắp xếp\*/

void array\_unique(int\* array, size\_t size){

    size\_t end = 0;

    for (size\_t i = 1; i < size; ++i) {

        if (array[i] != array[end]) {

            ++end;

            array[end] = array[i];

        }

    }

    //Liệt kê các phần tử xuất hiện một lần trong mảng C

    show\_array(array, end+1);

}

int main(void){

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    int array2[] = { 7, 2, 6, 7, 7 };

    int array3[] = { 7, 7, 7 };

    // Sắp xếp các mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    qsort(array2, SIZE\_OF\_ARRAY(array2), sizeof(int), compareIntAsc);

    qsort(array3, SIZE\_OF\_ARRAY(array3), sizeof(int), compareIntAsc);

    //Xoá phần tử trùng nhau và liệt kê các phần tử duy nhất

    array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    array\_unique(array2, SIZE\_OF\_ARRAY(array2));

    array\_unique(array3, SIZE\_OF\_ARRAY(array3));

    return 0;

}

2 4 6 7 8 9

2 6 7

7

### Liệt kê các phần tử xuất hiện một lần trong mảng

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm liệt kê các phần tử xuất hiện một lần trong mảng C\*/

void array\_unique(int\* array, size\_t size){

    for (size\_t i = 0; i < size; ++i) {

        size\_t count = 0;

        for (size\_t j = 0; j < size; ++j) {

            if (array[i] == array[j] &i!=j) {

                //Tìm thấy phần tử trùng nhau thì tăng biến đếm

               count += 1;

            }

        }

        /\*Nếu count vẫn bằng 0 thì phần tử đang kiểm tra

         chỉ xuất hiện 1 lần duy nhất trong mảng\*/

        if(count==0) printf("%d ",array[i]);

    }

    printf("\n");

}

int main(void){

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    int array2[] = { 7, 2, 6, 7, 7 };

    int array3[] = { 7, 7, 7 };

    //Xoá phần tử trùng nhau và liệt kê các phần tử xuất hiện một lần trong kết quả

    array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    array\_unique(array2, SIZE\_OF\_ARRAY(array2));

    array\_unique(array3, SIZE\_OF\_ARRAY(array3));

    return 0;

}

2 6 4 9 8

2 6

### Liệt kê các phần tử xuất hiện nhiều hơn 1 lần trong mảng

VD26

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Thay đổi hàm kiểm tra phần tử trùng nhau ở phương pháp một\*/

/\*Và tạo hàm lấy phần tử trùng trong mảng C\*/

void take\_duplicate\_element(const int\* array, size\_t size){

  int result[100], count=0;

    for (size\_t i = 0; i < size - 1; ++i) {

        for (size\_t j = i + 1; j < size; ++j) {

            if (array[i] == array[j]) {

            result[count]=array[i];

            ++ count;

            }

        }

    }

    //liệt kê các phần tử xuất hiện nhiều hơn 1 lần trong mảng C

    show\_array(result, count);

}

int main(void){

    int array1[] = {1,2,3,3,2,5};

    int array2[] = {1,1,2,3,2,5,5,8,9,6};

    //liệt kê các phần tử xuất hiện nhiều hơn 1 lần trong mảng C

    take\_duplicate\_element(array1,SIZE\_OF\_ARRAY(array1));

    take\_duplicate\_element(array2,SIZE\_OF\_ARRAY(array2));

    return 0;

}

2 3

1 2 5

## Tím số nguyên tố trong mảng

### Tìm số nguyên tố trong mảng

VD27

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số nguyên tố trong C\*/

int check\_prime\_number(int n){

    //flag = 0 => không phải số nguyên tố

    //flag = 1 => số nguyên tố

    int flag = 1;

    if (n <2) return flag = 0; /\*Số nhỏ hơn 2 không phải số nguyên tố => trả về 0\*/

    /\*Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không\*/

    int i = 2;

    while(i <n){

        if( n%i==0 ) {

            flag = 0;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

/\*Tạo hàm in ra các số nguyên tố trong mảng C\*/

void show\_prime\_number\_from\_array(int array[], int length){

    int prime\_number[100];

    for (int i = 0; i < length; i++){

      //Kiểm tra phần tử là số nguyên tố hay không

      //Nếu là số nguyên tố thì in ra màn hình

      if (check\_prime\_number(array[i]) == 1)  printf("%d ",array[i]);

    }

}

int main(void){

    //Khởi tạo mảng in ra các số nguyên tố

    int array[] = {0,1,2,3,4,5,6,7,8,9,10};

    //in ra các số nguyên tố trong mảng C bằng hàm đã tạo

    show\_prime\_number\_from\_array(array,SIZE\_OF\_ARRAY(array));

    return 0;

}

2 3 5 7

### Tách số nguyên tố trong mảng

VD29

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm kiểm tra số nguyên tố trong C\*/

int check\_prime\_number(int n){

    //flag = 0 => không phải số nguyên tố

    //flag = 1 => số nguyên tố

    int flag = 1;

    if (n <2) return flag = 0; /\*Số nhỏ hơn 2 không phải số nguyên tố => trả về 0\*/

    /\*Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không\*/

    int i = 2;

    while(i <n){

        if( n%i==0 ) {

            flag = 0;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    //Khai báo mảng chứa kết quả

    int prime\_number\_array[100];

    //Khởi tạo mảng tách ra các số nguyên tố

    int array[] = {0,7,2,3,6,5};

    //Sử dụng macro ở trên để lấy độ dài mảng

    int length = SIZE\_OF\_ARRAY(array);

    int prime\_number\_count = 0;

    for (int i = 0; i < length; i++){

        //Kiểm tra phần tử là số âm hay dương

        //Và gán số tìm thấy vào các mảng âm dương tương ứng

        //Cũng như tăng dần số phần tử trong mảng kết quả

        if (check\_prime\_number(array[i]) == 1) {

          prime\_number\_array[prime\_number\_count]=array[i];

          ++ prime\_number\_count;

        }

    }

    //In ra mảng số nguyên tố đã tách ra

    show\_array(prime\_number\_array, prime\_number\_count);

    return 0;

}

7 2 3 5

### Sắp xếp các số nguyên tố trong mảng

#include <stdio.h>

#include <stdlib.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm so sánh giảm dần sử dụng trong hàm sắp xếp qsort\*/

int compareIntDesc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return bNum - aNum;

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Tạo hàm kiểm tra số nguyên tố trong C\*/

int check\_prime\_number(int n){

    //flag = 0 => không phải số nguyên tố

    //flag = 1 => số nguyên tố

    int flag = 1;

    if (n <2) return flag = 0; /\*Số nhỏ hơn 2 không phải số nguyên tố => trả về 0\*/

    /\*Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không\*/

    int i = 2;

    while(i <n){

        if( n%i==0 ) {

            flag = 0;

            break; /\*Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    //Khai báo mảng chứa kết quả

    int prime\_number\_array[100];

    //Khởi tạo mảng tách ra các số nguyên tố

    int array[] = {0,7,2,3,6,5};

    //Sử dụng macro ở trên để lấy độ dài mảng

    int length = SIZE\_OF\_ARRAY(array);

    int prime\_number\_count = 0;

    for (int i = 0; i < length; i++){

        //Kiểm tra phần tử là số âm hay dương

        //Và gán số tìm thấy vào các mảng âm dương tương ứng

        //Cũng như tăng dần số phần tử trong mảng kết quả

        if (check\_prime\_number(array[i]) == 1) {

          prime\_number\_array[prime\_number\_count]=array[i];

          ++ prime\_number\_count;

        }

    }

    //In ra mảng số nguyên tố đã tách ra

    printf("Cac so nguyen to: ");

    show\_array(prime\_number\_array, prime\_number\_count);

    //Sắp xếp các số nguyên tố theo thứ tự tăng dần bằng hàm qsort

    qsort(prime\_number\_array, prime\_number\_count, sizeof(int), compareIntAsc);

    printf("Sap xep tang dan: ");

    show\_array(prime\_number\_array, prime\_number\_count);

    //Sắp xếp các số nguyên tố theo thứ tự giảm dần bằng hàm qsort

    qsort(prime\_number\_array, prime\_number\_count, sizeof(int), compareIntDesc);

    printf("Sap xep giam dan: ");

    show\_array(prime\_number\_array, prime\_number\_count);

    return 0;

}

Cac so nguyen to: 7 2 3 5

Sap xep tang dan: 2 3 5 7

Sap xep giam dan: 7 5 3 2

## Tím số chính phương trong mảng

### Tìm và đếm số chính phương trong mảng

#include <stdio.h>

#include <math.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chính phương trong C\*/

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int array[] = {5, 4, 16, 2, 8, 7, 3, 9};

    int length= SIZE\_OF\_ARRAY(array);

    int count = 0;

    /\*Tìm và đếm số chính phương trong mảng C\*/

    printf("Cac so chinh phuong trong mang: ");

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("%d ", array[i]);

            ++count;

        }

    }

    printf("\nSo so chinh phuong trong mang:%d so", count);

    return 0;

}

Cac so chinh phuong trong mang: 4 16 9

So so chinh phuong trong mang:3 so

### Tính tổng các số chính phương trong mảng

#include <stdio.h>

#include <math.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chính phương trong C\*/

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int array[] = {5, 4, 16, 2, 8, 7, 3, 9};

    int length= SIZE\_OF\_ARRAY(array);

    int total = 0;

    /\*Tìm và đếm số chính phương trong mảng C\*/

    printf("Cac so chinh phuong trong mang: ");

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("%d ", array[i]);

            total += array[i];

        }

    }

    printf("\nTong cac so chinh phuong trong mang: %d", total);

    return 0;

}

Cac so chinh phuong trong mang: 4 16 9

Tong cac so chinh phuong trong mang: 29

### Tính trung bình cộng các số chính phương trong mảng

#include <stdio.h>

#include <math.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chính phương trong C\*/

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int array[] = {5, 4, 16, 2, 8, 7, 3, 9};

    int length= SIZE\_OF\_ARRAY(array);

    int total = 0, count= 0;

    /\*Tìm và đếm số chính phương trong mảng C\*/

    printf("Cac so chinh phuong trong mang: ");

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("%d ", array[i]);

            total += array[i];

            ++count;

        }

    }

    printf("\nTrung binh cong cac so chinh phuong trong mang: %.2f", (float)total/count);

    return 0;

}

Cac so chinh phuong trong mang: 4 16 9

Trung binh cong cac so chinh phuong trong mang: 9.67

### Tìm số chính phương đầu tiên và cuối cùng có trong mảng

#include <stdio.h>

#include <math.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chính phương trong C\*/

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int array[] = {5, 4, 16, 2, 8, 7, 3, 9};

    int length= SIZE\_OF\_ARRAY(array);

    /\*tìm số chính phương đầu tiên có trong mảng\*/

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("So chinh phuong dau tien: %d \n", array[i]);

            break;

        }

    }

    /\*tìm số chính phương cuối cùng có trong mảng\*/

    for (size\_t i = length-1; i >= 0; --i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("So chinh phuong cuoi cung: %d ", array[i]);

            break;

        }

    }

    return 0;

}

So chinh phuong dau tien: 4

So chinh phuong cuoi cung: 9

### Tìm số chính phương lớn nhất và nhỏ nhất trong mảng

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm so sánh giảm dần sử dụng trong hàm qsort\*/

int compareIntDesc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return bNum - aNum;

}

/\*Tạo hàm kiểm tra số chính phương trong C\*/

int find\_square\_number(int n){

    //flag = 1 => số chính phương

    //flag = 0 => không phải số chính phương

    int flag = 0;

    //Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    int i = 0;

    while(i <= n){

        if( pow( i, 2) == n ) {

            flag = 1;

            break; /\*Chỉ cần tìm thấy 1 số là đủ và thoát vòng lặp\*/

        }

        i++;

    }

    return flag;

}

int main(void){

    int array[] = {5, 4, 16, 2, 8, 7, 3, 9};

    int length= SIZE\_OF\_ARRAY(array);

    /\*Sử dụng hàm qsort để sắp xếp mảng giảm dần\*/

    qsort(array, length, sizeof(int), compareIntDesc);

    /\*Tìm số chính phương đầu tiên có trong mảng

      Cũng chính là số chính phương lớn nhất trong mảng

    \*/

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("So chinh phuong lon nhat: %d \n", array[i]);

            break;

        }

    }

    /\*Tìm số chính phương cuối cùng có trong mảng

      Cũng chính là số chính phương nhỏ nhất trong mảng

    \*/

    for (size\_t i = length-1; i >= 0; --i) {

        int x = array[i];

        int check = find\_square\_number(x);

        if (check == 1) { //Tìm thấy số chính phương

            printf("So chinh phuong nho nhat: %d ", array[i]);

            break;

        }

    }

    return 0;

}

So chinh phuong lon nhat: 16

So chinh phuong nho nhat: 4

## Tìm phần tử trong mảng

### Tìm kiếm một phần tử trong mảng

#include <stdio.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)  (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm in phần tử trong mảng\*/

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

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

  printf("\n");

}

/\*Tạo hàm tìm vị trí phần tử trong mảng C\*/

void find\_element\_index(const int\* array, size\_t size, int x){

    int result[100], count = 0;

    for (size\_t i = 0; i < size - 1; ++i) {

        if (array[i] == x) {

            result[count] = i;

            ++count;

        }

    }

    //liệt kê các vị trí phần tử xuất hiện trong mảng

    printf("Phan tu %d xuat hien tai cac vi tri sau:\n", x);

    show\_array(result, count);

}

int main(void){

    int array[] = {1,1,2,3,2,5,5,8,9,6};

    //tìm vị trí phần tử trong mảng C

    find\_element\_index(array,SIZE\_OF\_ARRAY(array),3);

    find\_element\_index(array,SIZE\_OF\_ARRAY(array),5);

    return 0;

}

Phan tu 3 xuat hien tai cac vi tri sau:

3

Phan tu 5 xuat hien tai cac vi tri sau:

5 6

### Tìm phần tử xuất hiện nhiều nhất trong mảng

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

/\*Tạo macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) của mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array)    (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b)

{

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

/\*Biến tấu hàm đếm số lần xuất hiện của các phần tử

và tạo hàm tìm phần tử xuất hiện nhiều nhất trong mảng C\*/

void array\_unique(int\* array, size\_t size){

    //Khởi tạo mảng max chứa kết quả

    //max[0] : phần tử xuất hiện nhiều nhất

    //max[1] : số lần xuất hiện của phần tử đó

    int max[] ={0,0};

    //Sử dụng vòng lặp for để lọc ra các phần tử xuất hiện nhiều hơn 1 lần

    //So sánh số lần xuất hiện và thay đổi max khi cần.

    int count =1;

    for (size\_t i = size-1; i > 0; --i) {

        if (array[i] == array[i-1]) ++count; //Thấy phần tử trùng nhau thì tiếp tục đếm

        else{

             //So sánh số lần xuất hiện với max[1]

             if (max[1] < count){

                  //Nếu tìm thấy phần tử xuất hiện nhiều hơn thì gán phần tử vào max[0]

                  //Và gán số lần xuất hiện vào max[1]

                  max[0] = array[i];

                  max[1] = count;

             }

             count = 1;

        }

    }

    printf("Phan tu %d xuat hien nhieu nhat trong mang voi %d lan\n", max[0], max[1]);

}

int main(void){

    int array1[] = { 7, 2, 6, 7, 4, 9, 8 };

    // Sắp xếp các mảng theo thứ tự tăng dần

    qsort(array1, SIZE\_OF\_ARRAY(array1), sizeof(int), compareIntAsc);

    //Đếm số lần xuất hiện của các phần tử trong mảng C

    array\_unique(array1, SIZE\_OF\_ARRAY(array1));

    return 0;

}

Phan tu 7 xuat hien nhieu nhat trong mang voi 2 lan

### Tìm phần tử chẵn nhỏ nhất trong mảng

#include <stdio.h>

#include <stdlib.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chẵn lẻ trong C\*/

int check\_odd\_even(int n){

    //flag = 1 => số lẻ

    //flag = 0 => số chẵn

    int flag = 1;

    if( n % 2 == 0 ) flag= 0;

    return flag;

}

/\*Tạo hàm so sánh tăng dần sử dụng trong hàm qsort\*/

int compareIntAsc(const void\* a, const void\* b){

    int aNum = \*(int\*)a;

    int bNum = \*(int\*)b;

    return aNum - bNum;

}

int main(void){

    int array[] = {5, 4, 7, 2, 8, 7, 3};

    int length= SIZE\_OF\_ARRAY(array);

    /\*Sử dụng hàm qsort để sắp xếp mảng giảm dần\*/

    qsort(array, length, sizeof(int), compareIntAsc);

    /\*Phần tử phần tử chẵn nhỏ nhất trong mảng chính là phần tử chẵn đầu tiên\*/

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = check\_odd\_even(x);

        if (check == 0) { //Tìm thấy phần tử phần tử chẵn nhỏ nhất

            printf("Phan tu chan nho nhat trong mang= %d\n", array[i]);

            break;

        }

    }

    /\*Phần tử phần tử lẻ nhỏ nhất trong mảng chính là phần tử lẻ đầu tiên\*/

    for (size\_t i = 0; i < length; ++i) {

        int x = array[i];

        int check = check\_odd\_even(x);

        if (check == 1) { //Tìm thấy phần tử phần tử lẻ nhỏ nhất

            printf("Phan tu le nho nhat trong mang= %d\n", array[i]);

            break;

        }

    }

    return 0;

}

Phan tu chan nho nhat trong mang= 2

Phan tu le nho nhat trong mang= 3

### Tìm vị trí cuối cùng của phần tử x trong mảng

#include <stdio.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm tìm vị trí cuối cùng của phần tử x trong mảng C\*/

int find\_last\_x(int array[], size\_t n, int x){

    for (size\_t i = n-1; i >=0; i --){

        if (array[i]==x){

            return i;

        }

    }

}

int main(void){

    int array[] = {5, 4, 6, 7, 2, 8, 7, 3};

    int x;

    //Tìm độ dài (số phần tử ) của mảng

    int n= SIZE\_OF\_ARRAY(array);

    //Tìm vị trí cuối cùng của phần tử x trong mảng C

    x = 7;

    int k1 = find\_last\_x(array, n , x);

    printf("Vi tri cuoi cung cua so %d trong mang array la %d\n", x,k1);

    x = 4;

    int k2 = find\_last\_x(array, n , x);

    printf("Vi tri cuoi cung cua so %d trong mang array la %d\n", x,k2);

    return 0;

}

Vi tri cuoi cung cua so 7 trong mang array1 la 6

Vi tri cuoi cung cua so 4 trong mang array2 la 1

### Tìm số chẵn lẻ cuối cùng trong mảng

#include <stdio.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

/\*Tạo hàm kiểm tra số chẵn lẻ\*/

int check\_odd\_even(int n){

    //flag = 1 => số lẻ

    //flag = 0 => số chẵn

    int flag = 1;

    if( n % 2 == 0 ) flag= 0;

    return flag;

}

/\*Tạo hàm tìm số chẵn cuối cùng trong mảng C\*/

int find\_last\_odd(int array[], size\_t n){

    for (size\_t i = n-1; i >=0; i --){

        int x = array[i];

        if (check\_odd\_even(x)==0){

            return x;

        }

    }

}

/\*Tạo hàm tìm số lẻ cuối cùng trong mảng C\*/

int find\_last\_even(int array[], size\_t n){

    for (size\_t i = n-1; i >=0; i --){

        int x = array[i];

        if (check\_odd\_even(x)==1){

            return x;

        }

    }

}

int main(void){

    int array[] = {5, 4, 6, 7, 2, 8, 7, 3};

    //Tìm độ dài (số phần tử ) của mảng

    int n= SIZE\_OF\_ARRAY(array), k;

    //Tìm số chẵn cuối cùng trong mảng C

    k = find\_last\_odd(array, n);

    printf("So chan cuoi cung trong mang la %d\n", k);

    //Tìm số lẻ cuối cùng trong mảng C

    k = find\_last\_even(array, n);

    printf("So le cuoi cung trong mang la %d\n", k);

    return 0;

}

So chan cuoi cung trong mang la 8

So le cuoi cung trong mang la 3

### Tìm số âm dương cuối cùng trong mảng

#include <stdio.h>

/\*Định nghĩa macro SIZE\_OF\_ARRAY để lấy độ dài (số phần tử) trong mảng chỉ định\*/

#define SIZE\_OF\_ARRAY(array) (sizeof(array)/sizeof(array[0]))

int main(void){

    int array[] = {5, 4, -6, 7, -2, 8, 7, 3};

    //Tìm độ dài (số phần tử ) của mảng

    int n= SIZE\_OF\_ARRAY(array);

    //Tìm số dương cuối cùng trong mảng C

    for (size\_t i = n-1; i >=0; i --){

        if (array[i]>0){

             printf("So duong cuoi cung trong mang la %d\n", array[i]);

             break;

        }

    }

    //Tìm số âm cuối cùng trong mảng C

    for (size\_t i = n-1; i >=0; i --){

        if (array[i]<0){

             printf("So am cuoi cung trong mang la %d\n", array[i]);

             break;

        }

    }

    return 0;

}

So duong cuoi cung trong mang la 3

So am cuoi cung trong mang la -2

## Mảng chuỗi 2 chiều

Mảng chuỗi 2 chiều trong c, hay còn gọi là mảng ký tự 2 chiều trong c hoặc là mảng char 2 chiều trong c, là loại mảng 2 chiều với các phần tử của mảng đều có chung kiểu char, và được sử dụng để lưu giữ nhiều chuỗi ký tự cùng lúc trong cùng một mảng.

### Khởi tạo và truy cập:

VD

#include <stdio.h>

int main(void)

{

    char s[][4] = {"ABC", "DE", "FGH"};

    printf("%s\n", s[0]);

    printf("%s\n", s[1]);

    printf("%s\n", s[2]);

    return 0;

}

ABC

DE

FGH

### Lấy kích thước

#include <stdio.h>

int main(void){

    char s[][4] = {"ABC", "DE", "FGH"};

    int size = sizeof s;

    printf("%d",size);

}

//12

### Lấy độ dài

#include <stdio.h>

int main(void){

    char fruits[][10] = {"Apple", "Kiwi", "Lemon", "Melon", "Orange"};

     //Lấy độ dài của mảng 2 chiều

    int y = sizeof(fruits) / sizeof(fruits[0]);

    int x = sizeof(fruits[0]) / sizeof(fruits[0][0]);

    printf("Do dai mang 2 chieu: y = %d , x = %d",y, x);

}

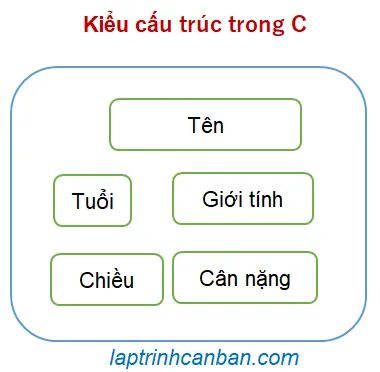
//Do dai mang 2 chieu: y = 5 , x = 10

# 6. Kiểu cấu trúc

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/kieu-cau-truc-trong-c/kieu-cau-truc-trong-c/>

## Khái niệm:

Kiểu cấu trúc trong C, hay còn gọi là kiểu struct trong C là một tập hợp các thuộc tính liên quan tới cùng một đối tượng.



Các thuộc tính tạo nên một cấu trúc được gọi là các **thành viên** (tiếng Anh: member) của cấu trúc đó.

Hàm không thể được sử dụng làm thành viên của một cấu trúc.

Từ một cấu trúc, chúng ta có thể tạo ra vô vàn các thực thể (tiếng Anh: instance) có cấu trúc giống nhau.

## Khái báo

### Cách 1: lệnh struct

struct name{

type1 member1;

type2 member2;

type3 member3;

...

};

Trong đó name là tên của cấu trúc cần khai báo, và các cặp type và member là kiểu và tên của các thành viên trong cấu trúc đó.

Ví dụ:

struct people1{

    int old; // tuổi

    char \*name; // tên

    int height; // chiều cao

    char \*sex; // giới tính

};

### Cách 2: sử dụng typedef

Chúng ta sử dụng cú pháp tổng quát của typedef struct trong c để định nghĩa lại tên của kiểu struct như sau:

typedef struct old\_struct\_name{

type1 member1;

type2 member2;

type3 member3;

...

}new\_struct\_name;

Tuy nhiên trong thực tế thì chúng ta hay lược bỏ đi old\_struct\_name và khai báo cấu trúc với cú pháp sau đây:

typedef struct{

type1 member1;

type2 member2;

type3 member3;

...

}struct\_name;

VD:

typedef struct{

    int old; // tuổi

    char \*name; // tên

    int height; // chiều cao

    char \*sex; // giới tính

}people2;

## Khởi tạo:

### Cách 1: Với thực thể được tạo bởi cách 1 (sử dụng struct)

struct struct\_name instance\_name;

VD:

struct people1{

    int old; // tuổi

    char \*name; // tên

    int height; // chiều cao

    char \*sex; // giới tính

};

int main(void){

    struct people1 BaKien;

}

VD: Gán giá trị ban đầu

int main(void){

    struct people1 BaKien = {40, "BaKien",160, "male" };

}

### Cách 2: Với thực thể được tạo bởi cách 2 (sử dụng typedef)

struct\_name instance\_name;

VD:

typedef struct{

    int old; // tuổi

    char \*name; // tên

    int height; // chiều cao

    char \*sex; // giới tính

}people2;

int main(void){

    people2 ChiPheo;

}

VD: Gán giá trị ban đầu

int main(void){

    people2 ChiPheo = {20, "ChiPheo",180, "male" };

}

## Truy cập vào các thành viên trong cấu trúc

Chúng ta có 2 phương pháp để truy cập vào thành viên trong một thực thể như sau:

1. Sử dụng dấu chấm . trong các trường hợp sử dụng biến thông thường
2. Sử dụng dấu mũi tên -> trong trường hợp sử dụng biến con trỏ.

VD:

int main(void){

    struct people1 BaKien;

    BaKien.old = 40;

    BaKien.name = "BaKien";

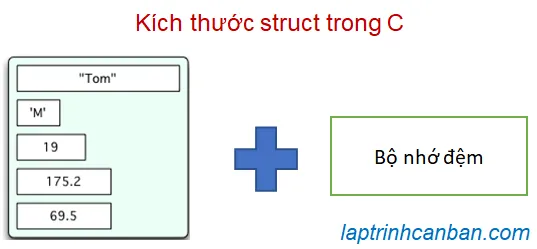
    BaKien.height = 160;

    BaKien.sex = "male";

}

## Kích thước struct

Kích thước struct trong C được tính bởi đơn vị byte, và kích thước của nó được tính bằng tổng kích thước của tất cả các thành viên trong nó, cộng thêm một bộ nhớ đệm (padding memory) ở giữa các thành viên nhằm tăng tốc độ truy cập của máy tính.



Nguyên nhân là do các thành viên của struct được bố trí ở những vị trí thuận tiện cho việc truy cập nhằm tăng tốc độ truy cập của máy tính. Kết quả là, một khoảng trống (được gọi là đệm, tiếng anh là padding) được tạo ra giữa các thành viên và làm kích thước bộ nhớ tổng thể tăng lên.

### Lấy kích thước struct trong C | toán tử sizeof()

VD:

#include <stdio.h>

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

int main(void){

  person\_t p;

  printf("name=%d, sex=%d, age=%d, height=%d, weight=%d\n",

         sizeof(p.name), sizeof(p.sex), sizeof(p.age), sizeof(p.height), sizeof(p.weight));

  printf("person\_t=%d\n", sizeof(p));

}

name=20, sex=1, age=4, height=8, weight=8

person\_t=48

Bạn có thể thấy rõ tổng kích thước của tất cả các thành viên trong struct là **41**, tuy nhiên kích thước thực của struct lại là **48**. Lý do là vì một khoảng **bộ nhớ trống** (padding memory) đã được thêm vào giữa các thành viên, nhằm tăng tốc độ truy cập của máy tính tới các thành viên trong struct. Kết quả, kích thước thực của struct mới lớn hơn tổng kích thước của các thành viên như trên.

## Con trỏ cấu trúc

### Sử dụng tới thành viên

Sử dụng dấu -> để trỏ tới thành viên từ các thực thể.

instance\_pointer -> member

Ví dụ:

#include <stdio.h>

//Khai báo cấu trúc

typedef struct person {

    char \*name;

    char sex;

    int age;

    char \*add;

    char \*job;

} person2;

int main(void) {

    //Khai báo thực thể kiyoshi, và con trỏ tới thực thể này

    person2 kiyoshi, \*p1;

    // Gán địa chỉ của thực thể vào con trỏ

    p1 = &kiyoshi;

    //Truy cập và gán giá trị vào thành viên trong con trỏ cấu trúc

    p1->name = "Kiyoshi";

    p1->sex = 'M';

    p1->age = 20;

    p1->add = "Tokyo";

    p1->job = "BrSE";

    printf("kiyoshi= %s %c %d %s %s\n",

        kiyoshi.name, kiyoshi.sex, kiyoshi.age, kiyoshi.add, kiyoshi.job);

     //Khai báo thực thể honda, và con trỏ tới thực thể này

    person2 honda, \*p2;

    // Gán địa chỉ của thực thể vào con trỏ

    p2 = &honda;

    //Gán con trỏ của thực thể kiyoshi vào con trỏ của thực thể honda

    //Qua đó tiến hành copy thực thể kiyoshi vào thực thể honda

    \*p2 = \*p1;

    printf("honda= %s %c %d %s %s\n",

        honda.name, honda.sex, honda.age, honda.add, honda.job);

    return 0;

}

kiyoshi= Kiyoshi M 20 Tokyo BrSE

honda= Kiyoshi M 20 Tokyo BrSE

### Sử dụng con trỏ cấu trúc trong hàm

Lợi ích của việc truyền con trỏ cấu trúc vào hàm thay vì truyền cấu trúc

1. Một là mất ít thời gian hơn để chỉ truyền giá trị con trỏ so với giá trị của toàn bộ cấu trúc. Điều này rất quan trọng đối với việc sử dụng hiệu quả bộ nhớ và thời gian thực thi
2. Hai là dễ dàng thay đổi nội dung của cấu trúc từ bên ngoài hàm

VD:

#include <stdio.h>

//Khai báo cấu trúc biểu diễn số phức

typedef struct {

    double re;    /\* Phần thực \*/

    double im;    /\* Phần ảo \*/

} complex\_t;

//Tạo hàm tính tổng các cấu trúc thôg qua con trỏ cấu trúc

void addComplexPtr(complex\_t \*a, complex\_t \*b, complex\_t \*c)

{

    (\*c).re = (\*a).re + (\*b).re;

    (\*c).im = (\*a).im + (\*b).im;

}

//Tạo hàm nhận và in thực thể của cấu trúc

void printComplex(complex\_t c)

{

    printf("%f + %f i\n", c.re, c.im);

}

int main(void){

    //Khởi tạo các thực thể của cấu trúc complex\_t

    complex\_t x = {1.2, 3.4};

    complex\_t y = {5.6, 7.8};

    complex\_t z;

    //Khởi tạo các con trỏ chứa địa chỉ các thực thể của cấu trúc

    complex\_t \*p1 = &x;

    complex\_t \*p2 = &y;

    complex\_t \*p3 = &z;

    addComplexPtr(p1, p2, p3);//addComplexPtr(&x, &y, &z);

    printComplex(z);

    return 0;

}

6.800000 + 11.200000 i

## Mảng cấu trúc

**Mảng cấu trúc trong C** là mảng chứa các thực thể được tạo ra từ một kiểu cấu trúc bên trong nó. Do thực thể tạo ra từ cấu trúc cũng là một loại giá trị, nên chúng ta cũng có thể lưu trữ chúng như là phần tử trong cùng một mảng. Và các phần tử trong mảng cấu trúc đều có chung kiểu là kiểu cấu trúc đã sử dụng để tạo nên thực thể.

Mảng cấu trúc trong C có đặc tính của cả kiểu mảng và kiểu cấu trúc, do đó chúng ta có thể truy cập vào các thực thể trong mảng cấu trúc thông qua index, cũng như là truy cập vào các thành viên của từng thực thể theo cách thông thường.

### Khai báo

struct\_name array\_name[length];

VD

//Khai báo kiểu cấu trúc

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

main(void){

    //Khai báo mảng cấu trúc chứa 3 thực thể của kiểu cấu trúc

    person\_t p[3];

}

### Khởi tạo và truy cập

Tương tự như mảng.

VD:

#include <stdio.h>

//Định nghĩa số thực thể trong mảng cấu trúc

#define PERSON\_NUM 5

//Khai báo kiểu cấu trúc

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

main(void){

    //Khởi tạo mảng cấu trúc với 5 thực thể được tạo ra từ cấu trúc

    person\_t p[PERSON\_NUM] = {{"Bob",      'M', 19, 165.4, 72.5},

                              {"Alice",    'F', 19, 161.7, 44.2},

                              {"Tom",      'M', 20, 175.2, 66.3},

                              {"Stefany",  'F', 18, 159.3, 48.5},

                              {"Leonardo", 'M', 19, 172.8, 67.2}};

    //Khai báo các biến tổng và trung bình của chiều cao và cân nặng

    double height\_sum, weight\_sum, height\_ave, weight\_ave;

    height\_sum = weight\_sum = 0.0;

    //Sử dụng vòng lặp để tính tổng chiều cao và cân nặng từ các thực thể

    for (short i = 0; i < PERSON\_NUM; i++) {

        height\_sum += p[i].height;

        weight\_sum += p[i].weight;

    }

    //Từ tổng chiều cao và cân nặng tính ra giá trị trung bình

    height\_ave = height\_sum / PERSON\_NUM;

    weight\_ave = weight\_sum / PERSON\_NUM;

    printf("Chieu cao trung binh = %f\n", height\_ave);

    printf("Can nang trung binh = %f\n", weight\_ave);

    return 0;

}

Chieu cao trung binh = 166.880000

Can nang trung binh = 59.740000

## Truyền cấu trúc cho hàm

VD1:

#include <stdio.h>

//Khai báo cấu trúc biểu diễn số phức

typedef struct {

    double re;    /\* Phần thực \*/

    double im;    /\* Phần ảo \*/

} complex\_t;

//Tạo hàm nhận và in thực thể của cấu trúc

void printComplex(complex\_t c)

{

    printf("%f + %f i\n", c.re, c.im);

}

int main(void){

    //Tạo một thực thể từ cấu trúc

    complex\_t c = {1.2, 3.4};

    //Truyền thực thể vào hàm

    printComplex(c);

}

1.200000 + 3.400000 i

## Trả về cấu trúc từ hàm

VD2

#include <stdio.h>

typedef struct {

    double re;    /\* Phần thực \*/

    double im;    /\* Phần ảo \*/

} complex\_t;

//Tạo hàm nhận, tính toán và trả về thực thể của cấu trúc

complex\_t addComplex(complex\_t a, complex\_t b){

    complex\_t c;

    c.re = a.re + b.re;

    c.im = a.im + b.im;

    return c;

}

//Tạo hàm nhận và in thực thể của cấu trúc

void printComplex(complex\_t c){

    printf("%f + %f i\n", c.re, c.im);

}

int main(void){

    complex\_t a = {1.2, 3.4};

    complex\_t b = {5.6, 7.8};

    complex\_t c;

    c = addComplex(a, b);

    printComplex(c);

    return 0;

}

6.800000 + 11.200000 i

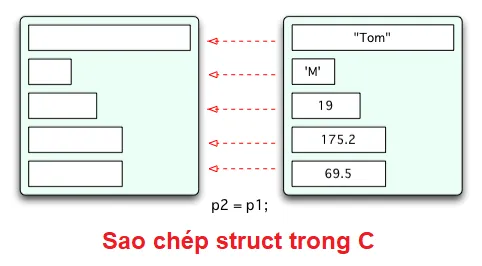
## Sao chép struct

### Sao chép struct trong C bằng toán tử bằng

Để sao chép (copy) toàn bộ nội dung của một thực thể của một cấu trúc sang một thực thể khác cùng kiểu, chúng ta có thể sử dụng toán tử bằng để gán giá trị, với cú pháp sau đây:

des\_instance = src\_instance;

Trong đó src\_instance là thực thể nguồn cần copy, và des\_instance là thực thể đích cần dán vào.



VD3

#include <stdio.h>

typedef struct {

    char name[20];

    char sex[10];

    int age;

    double height;

    double weight;

} person\_1;

int main(void)

{

    person\_1 p1 = {"Tom", "M", 19, 175.2, 69.5};

    person\_1 p2;

    p2 = p1; /\*Sao chép struct p1 sang p2 \*/

    printf("p1: %s %s %d %.1f %.1f\n", p1.name, p1.sex, p1.age, p1.height, p1.weight);

    printf("p2: %s %s %d %.1f %.1f\n", p2.name, p2.sex, p2.age, p2.height, p2.weight);

    return 0;

}

p1: Tom M 19 175.2 69.5

p2: Tom M 19 175.2 69.5

### Sao chép struct trong C bằng con trỏ

Để làm được điều này thì chúng ta cần tạo ra con trỏ nguồn chứa địa chỉ của thực thể nguồn, và con trỏ đích để chứa địa chỉ của thực thể đích. Sau đó thì gán giá trị của con trỏ nguồn cho con trỏ đích là xong.

VD4

#include <stdio.h>

typedef struct person {

    char \*name;

    char sex;

    int age;

    char \*add;

    char \*job;

} person2;

int main(void) {

    person2 kiyoshi = {"Kiyoshi", 'm', 30, "Tokyo", "BrSE"};

    printf("p1= %s %d %s %s\n", kiyoshi.name, kiyoshi.age, kiyoshi.add, kiyoshi.job);

    //Khai báo con trỏ nguồn và gán địa chỉ thực thể nguồn

    person2 \*p1;

    p1 = &kiyoshi;

    //Khai báo con trỏ đích và gán địa chỉ thực thể đích

    person2 kiyoshi2, \*p2;

    p2 = &kiyoshi2;

    //Gán con trỏ nguồn vào con trỏ đích

    //Qua đó sao chép thực thể nguồn vào thực thể đích

    \*p2 = \*p1;

    printf("p2= %s %d %s %s", kiyoshi2.name, kiyoshi2.age, kiyoshi2.add, kiyoshi2.job);

    return 0;

}

p1= Kiyoshi 30 Tokyo BrSE

p2= Kiyoshi 30 Tokyo BrSE

## So sánh struct

### So sánh 2 struct trong C bằng cách so sánh các thành viên với nhau

Để so sánh 2 struct trong C xem chúng có giống nhau hay không, chúng ta sẽ tiến hành so sánh lần lượt giá trị của các thành viên của chúng. Nếu chỉ cần một cặp thành viên trong chúng có giá trị khác nhau thì chúng ta có thể kết luận là 2 struct này khác nhau. Ngược lại nếu tất cả các cặp giá trị giữa chúng đều giống nhau, điều đó chứng tỏ 2 struct này hoàn toàn giống nhau.

VD5

#include <stdio.h>

#include <string.h>

//Khai báo struct person

typedef struct person {

    char \*name;

    char sex;

    int age;

    char \*add;

    char \*job;

} person2;

//Tạo hàm so sánh 2 struct trong c

int compare\_struct(person2 \*p1, person2 \*p2) {

    // return 1: Khác nhau

    // return 0: Giống nhau

    //So sánh các thành viên thuộc kiểu số

    if(p1->sex != p2->sex || p1->age != p2->age) {

        return 1;

    }

    //So sánh các thành viên thuộc kiểu chuỗi bằng hàm strcmp

    if(strcmp(p1->name, p2->name) != 0) {

        return 1;

    }

    if(strcmp(p1->add, p2->add) != 0) {

        return 1;

    }

    if(strcmp(p1->job, p2->job) != 0) {

        return 1;

    }

    return 0;

}

int main(void) {

    person2 kiyoshi = {"kiyoshi", 'm', 30, "Tokyo", "BrSE"};

    //Tạo kiyoshi2 giống kiyoshi

    //bằng cách copy struct kiyoshi sang kiyoshi2 thông qua con trỏ

    person2 \*p1;

    p1 = &kiyoshi;

    person2 kiyoshi2, \*p2;

    p2 = &kiyoshi2;

    \*p2 = \*p1;

    //Tạo thực thể honda khác kiyoshi bằng cách khởi tạo giá trị mới

    person2 honda = {"honda", 'm', 30, "Tokyo", "BrSE"};

    // So sánh struct kiyoshi và kiyoshi2

    if(compare\_struct(&kiyoshi, &kiyoshi2) == 0) {

        printf("struct %s giong struct %s\n", "kiyoshi", "kiyoshi2");

    } else {

        printf("struct %s khac struct %s\n", "kiyoshi", "kiyoshi2");

    }

    // So sánh struct kiyoshi và honda

    if(compare\_struct(&kiyoshi, &honda) == 0) {

        printf("struct %s giong struct %s\n", "kiyoshi", "honda");

    } else {

        printf("struct %s khac struct %s\n", "kiyoshi", "honda");

    }

    return 0;

}

struct kiyoshi giong struct kiyoshi2

struct kiyoshi khac struct honda

### So sánh 2 struct trong C bằng cách so sánh kích thước 2 struct với nhau

Phương pháp thứ 2 để so sánh 2 struct trong C đó chính là so sánh kích thước chiếm trong bộ nhớ của 2 struct với nhau.

Ý tưởng ở đây rất đơn giản. Chúng ta lấy kích thước của 2 struct này và so sánh chúng. Nếu chúng có cùng kích thước (byte) chiếm trong bộ nhớ thì điều đó chứng tỏ là chúng giống nhau. Và ngược lại nếu kích thước của chúng khác nhau thì điều đó chứng tỏ là chúng khác nhau.

Để so sánh kích thước chiếm trong bộ nhớ của 2 struct, chúng ta sẽ dùng tới **hàm memcmp trong C**. Hàm memcmp thường được sử dụng để so sánh quan hệ lớn nhỏ giữa 2 vùng trong bộ nhớ, tuy nhiên bằng cách kiểm tra giá trị trả về của nó là 0 hay là một số khác 0 thì chúng ta cũng có thể so sánh xem 2 struct trong C có giống nhau hay không.

VD6

#include <stdio.h>

#include <string.h>

//Khai báo struct person

typedef struct person {

    char \*name;

    char sex;

    int age;

    char \*add;

    char \*job;

} person2;

int main(void) {

    person2 kiyoshi = {"kiyoshi", 'm', 30, "Tokyo", "BrSE"};

    //Tạo kiyoshi2 giống kiyoshi

    //bằng cách copy struct kiyoshi sang kiyoshi2 thông qua con trỏ

    person2 \*p1;

    p1 = &kiyoshi;

    person2 kiyoshi2, \*p2;

    p2 = &kiyoshi2;

    \*p2 = \*p1;

    //Tạo thực thể honda khác kiyoshi bằng cách khởi tạo giá trị mới

    person2 honda = {"sato", 'm', 30, "Tokyo", "BrSE"};

    // So sánh struct kiyoshi và kiyoshi2

    if(memcmp(&kiyoshi, &kiyoshi2, sizeof(person2)) == 0) {

        printf("struct %s giong struct %s\n", "kiyoshi", "kiyoshi2");

    } else {

        printf("struct %s khac struct %s\n", "kiyoshi", "kiyoshi2");

    }

    // So sánh struct kiyoshi và honda

    if(memcmp(&kiyoshi, &honda, sizeof(person2)) == 0) {

        printf("struct %s giong struct %s\n", "kiyoshi", "honda");

    } else {

        printf("struct %s khac struct %s\n", "kiyoshi", "honda");

    }

    return 0;

}

struct kiyoshi giong struct kiyoshi2

struct kiyoshi khac struct honda

## Xắp xếp struct

Để sắp xếp tăng dần các struct là phần tử của mảng cấu trúc trong C, chúng ta so sánh giá trị của các thành viên trong phần tử của mảng struct và tiến hành hoán đổi phần tử khi tìm thấy phần tử phía sau có giá trị thành viên lớn hơn phần tử đứng trước nó.

Ngược lại thì để sắp xếp giảm dần các struct là phần tử của mảng cấu trúc trong C, chúng ta so sánh giá trị của các thành viên trong phần tử của mảng struct và tiến hành hoán đổi phần tử khi tìm thấy phần tử phía sau có giá trị thành viênnhỏ hơn phần tử đứng trước nó.

VD7

#include <stdio.h>

#include <string.h>

/\*Định nghĩa macro SWAP để hoán đổi phần tử trong mảng chỉ định\*/

#define SWAP(type,x,y) do{type tmp = x; x = y; y = tmp;}while(0)

#define PERSON\_NUM 5

//Khai báo kiểu cấu trúc

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

//Tạo hàm in kiểu cấu trúc

void show\_struct(person\_t array[], int n){

    for (short i = 0; i < PERSON\_NUM; i++) {

      printf("\n%s %c %d %.1f %.1f",array[i].name,array[i].sex,array[i].age,array[i].height,array[i].weight);

   }

    printf("\n");

}

//Tạo hàm sắp xếp struct tăng dần theo chiều cao

void asc\_order\_height(person\_t array[], int n){

    //Tạo vòng lặp để sắp xếp struct

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            //Nếu tìm ra phần tử có thành viên height lớn hơn

            //thì hoán đổi với số đang xét

            if (array[i].height > array[j].height) SWAP(person\_t,array[i],array[j]);

        }

    }

}

//Tạo hàm sắp xếp struct giảm dần theo chiều cao

void desc\_order\_height(person\_t array[], int n){

    //Tạo vòng lặp để sắp xếp struct

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            //Nếu tìm ra phần tử có thành viên height nhỏ hơn

            // thì hoán đổi với số đang xét

            if (array[i].height < array[j].height) SWAP(person\_t,array[i],array[j]);

        }

    }

}

// Tạo hàm sắp xếp struct tăng dần theo tên

void asc\_order\_name(person\_t array[], int n){

    //Tạo vòng lặp để sắp xếp struct

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            //Nếu tìm ra phần tử có thành viên name lớn hơn

            // thì hoán đổi với số đang xét

            if (strcmp(array[i].name, array[j].name) > 0) SWAP(person\_t,array[i],array[j]);

        }

    }

}

// Tạo hàm sắp xếp struct giảm dần theo tên

void desc\_order\_name(person\_t array[], int n){

    //Tạo vòng lặp để sắp xếp struct

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            //Nếu tìm ra phần tử có thành viên name nhỏ hơn

            // thì hoán đổi với số đang xét

            if (strcmp(array[i].name, array[j].name) < 0) SWAP(person\_t,array[i],array[j]);

        }

    }

}

int main(void){

    //Khởi tạo mảng cấu trúc với 5 thực thể được tạo ra từ cấu trúc

    person\_t p[PERSON\_NUM] = {{"Bob", 'M', 19, 165.4, 72.5},

                              {"Ali", 'F', 19, 161.7, 44.2},

                              {"Tom", 'M', 20, 175.2, 66.3},

                              {"Ste", 'F', 18, 159.3, 48.5},

                              {"Leo", 'M', 19, 172.8, 67.2}};

    printf("Mang struct ban dau:");

    show\_struct(p,PERSON\_NUM);

    printf("\nSap xep theo chieu cao tang dan:");

    asc\_order\_height(p,PERSON\_NUM);

    show\_struct(p,PERSON\_NUM);

    printf("\nSap xep theo chieu cao giam dan:");

    desc\_order\_height(p,PERSON\_NUM);

    show\_struct(p,PERSON\_NUM);

    printf("\nSap xep theo ten tang dan:");

    asc\_order\_name(p,PERSON\_NUM);

    show\_struct(p,PERSON\_NUM);

    printf("\nSap xep theo ten giam dan:");

    desc\_order\_name(p,PERSON\_NUM);

    show\_struct(p,PERSON\_NUM);

}

Mang struct ban dau:

Bob M 19 165.4 72.5

Ali F 19 161.7 44.2

Tom M 20 175.2 66.3

Ste F 18 159.3 48.5

Leo M 19 172.8 67.2

Sap xep theo chieu cao tang dan:

Tom M 20 175.2 66.3

Leo M 19 172.8 67.2

Bob M 19 165.4 72.5

Ali F 19 161.7 44.2

Ste F 18 159.3 48.5

Sap xep theo chieu cao giam dan:

Ste F 18 159.3 48.5

Ali F 19 161.7 44.2

Bob M 19 165.4 72.5

Leo M 19 172.8 67.2

Tom M 20 175.2 66.3

Sap xep theo ten tang dan:

Ali F 19 161.7 44.2

Bob M 19 165.4 72.5

Leo M 19 172.8 67.2

Ste F 18 159.3 48.5

Tom M 20 175.2 66.3

Sap xep theo ten tang dan:

Tom M 20 175.2 66.3

Ste F 18 159.3 48.5

Leo M 19 172.8 67.2

Bob M 19 165.4 72.5

Ali F 19 161.7 44.2

# 7. 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

# 8. 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.

# Xử lý file

Trong thế giới máy tính, file là một loại “Tài liệu” nhằm lưu dữ dữ liệu và thông tin. Có rất nhiều kiểu file khác nhau nhằm lưu dữ các kiểu dữ liệu và thông tin khác nhau, ví dụ như file text, file Excel, file Json, file XML, hay là file CSV v.v…

Để có thể thao tác với các loại file này, người dùng thông thường cần các phần mềm chuyên dùng để mở, đọc ghi và lưu chúng. Tuy nhiên thì với các lập trình viên như chúng ta thì bằng cách sử dụng sử dụng ngôn ngữ C, chúng ta cũng có thể dễ dàng thực hiện các thao tác tương tự với file như vậy.

Cũng giống như mảng, chuỗi hay biến thì file trong C là một loại dữ liệu trong chương trình, và một file khi nhập vào chương trình cũng sẽ được lưu giữ tại một vùng nào đó trên bộ nhớ máy tính. Tuy nhiên khác với các loại dữ liệu khác có thể gán vào một biến để xử lý, thì để thao tác với file, chúng ta cần tạo ra một thực thể của kiểu cấu trúc FILE để chứa thông tin của file cần thao tác, sau đó sử dụng tới một con trỏ để chỉ đến vị trí của thực thể FILE chứa thông tin file đó trên bộ nhớ, qua đó thực hiện các thao tác với file thông qua các hàm có sẵn, với các chức năng cụ thể như đóng, mở, ghi hay lưu file.

Do vậy muốn thao tác được với file, bạn cần nắm vững các kiến thức cơ bản về con trỏ và kiểu cấu trúc trong C. Bạn có thể tham khảo các bài viết chi tiết về con trỏ trong chuyên đề Con trỏ trong C, cũng như về kiểu cấu trúc tại chuyên đề Kiểu cấu trúc trong C.

## Quy trình xử lý file

Quy trình xử lý file trong C sẽ gồm các bước như sau:



### Tạo con trỏ file

Để thao tác với file, trước tiên chúng ta cần phải tạo một con trỏ chỉ đến thực thể của kiểu cấu trúc FILE chứa thông tin của file đó trên bộ nhớ với cú pháp sau đây:

FILE \*fp;

Sau đó, bằng cách truy cập vào địa chỉ này thông qua con trỏ, chúng ta mới có thể thực hiện các thao tác với file.

### Mở file

khi mở file trong chương trình C, chúng ta cần phải sử dụng tới một số hàm chuyên dụng như fopen() hay fopen\_s().

Việc mở file và gán địa chỉ file vào con trỏ thường được tiến hành đồng thời. Ví dụ bạn có thể mở một file sample.txt bằng hàm fopen() và gán vào con trỏ file fp như sau:

fp = fopen("sample.txt", "r");

### Đọc dữ liệu từ file

Sau khi mở file, chúng ta đã có thể đọc các dữ liệu từ file vào trong chương trình để xử lý.

Có nhiều phương pháp để đọc dữ liệu từ một file. Ví dụ bạn có thể đọc từng dòng của file, hoặc đọc toàn bộ nội dung file. Nội dung file có thể được đọc dưới dạng từng ký tự, dạng chuỗi, hoặc dưới dạng mảng để có thể dễ dàng xử lý trong chương trình.

Trong C cũng cung cấp khá nhiều hàm có sẵn để đọc file như là fgets() hay fgetc() chẳng hạn.

Ngoài ra với từng định dạng file mà chúng ta sẽ có các phương pháp đọc file khác nhau trong C. Ví dụ như cùng sử dụng hàm fscanf() hoặc hàm sscanf() để đọc file, nhưng cú pháp đọc file txt sẽ khác cách đọc file CSV chẳng hạn.

Ví dụ cụ thể, sau khi mở file sample.txt ở trên, chúng ta có thể đọc từng dòng file đó bằng hàm fgets như sau:

VD

fp = fopen("sample.txt", "r");

int N = 256l

char str[N];

while(fgets(str, N , fp) != NULL) {

    printf("%s", str);

}

### Ghi file

Chúng ta có thể tạo một file mới rồi ghi nội dung vào file đó, hoặc là mở một file sẵn và ghi thêm nội dung vào đó.

Chúng ta có thể sử dụng các hàm có sẵn như fputc,fputs và fprintf để làm được việc này.

Ví dụ cụ thể, sau khi mở file sample.txt ở trên, chúng ta có thể ghi dòng chữ “Hello Vietnam!” vào file đó bằng hàm fprintf như sau:

fp = fopen("sample.txt", "r");

fprintf(fp, "%s", "Hello Vietnam!");

### Đóng file

Sau khi đã xử lý xong file trong chương trình, chúng ta cần phải đóng file đó lại. Việc đóng file sẽ giúp kết thúc phiên làm việc với file, và giải phóng bộ nhớ.

Nếu không đóng file thì file đó vẫn tồn tại trên bộ nhớ, dẫn đến xảy ra các sự cố về bộ nhớ trong chương trình.

Để đóng một file trong C, chúng ta cần dùng đến hàm fclose() với cú pháp sau đây:

fclose(fp);

Trong đó fp là con trỏ dùng để mở file.

### Xử lý lỗi khi thao tác với file

Chúng ta có thể sử dụng hàm fopen() để mở một file trong C, nhưng trong quá trình mở có thể gặp một số lỗi nào đó, dẫn dến việc không phải lúc nào bạn cũng có thể mở file đó thành công.

Sẽ tất nguy hiểm khi thực hiện quá trình thao tác với một tập tin mà không thể mở được, do đó thay vì để chương trình tiếp tục chạy như cũ thì chúng ta cần phải thiết kế các xử lý để chương trình tự xử lý lỗi, và chạy chương trình một cách chính xác.

Nếu hàm fopen() không thể mở file chính xác, nó sẽ trả về giá trị NULL. Chúng ta có thể sử dụng điều này để phán đoán khi nào mở file thất bại và xử lý lỗi khi cần.

Giả sử bạn đã mở một file sample.txt và gán vào con trỏ file fp như sau:

fp = fopen("sample.txt", "r");

Khi đó, nếu mở file thất bại thì giá trị của fp sẽ trở thành NULL, và chúng ta có thể viết xử lý tránh lỗi như sau:

if (fp == NULL) {

    //Nội dung xử lý khi lỗi mở file xảy ra

}

Về nội dung xử lý lỗi, thông thường chúng ta sẽ xuất một dòng thông báo “Đã xảy ra lỗi”, và sau đó buộc chương trình phải kết thúc.

Và để buộc chương trình phải kết thúc, chúng ta thường sử dụng tới hàm exit() trong header file stdlib.h nhằm cưỡng chế kết thúc chương trình.

exit(status);

Trong đó status là trạng thái thoát chương trình theo cách bình thường hay bất thường. Đối số status có thể chỉ định bằng một trong hai giá trị là 0 và 1, tương ứng với việc kết thúc bình thường và kết thúc bất thường.

Ngoài ra, chúng ta cũng có thể chỉ định giá trị của status là EXIT\_SUCCESS tương ứng với 0, và EXIT\_FAILURE tương ứng với 1.

Ví dụ, các cách viết sau đều OK khi sử dụng hàm exit() để kết thúc chương trình C.

// Kết thúc chương trình bình thường

exit(0);

exit(EXIT\_SUCCESS);

// Kết thúc chương trình bất thường

exit(1);

exit(EXIT\_FAILURE);

Lưu ý nếu bạn mở file bên trong hàm main(), thì thay vì dùng hàm exit() thì bạn cũng có thể dùng lệnh return như sau để xử lý lỗi khi mở file trong C:

//Mở file bằng hàm fopen, và trả về NULL nếu mở file thất bại.

fp = fopen(fname, "r");

if(fp == NULL) {

    printf("%s file not open!\n", fname);

    return -1;

}

Ứng dụng các kiến thức trên, chúng ta có thể viết xử lý lỗi khi mở file trong C như sau:

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

    //Khai báo con trỏ file

    FILE \* fp = NULL;

    fp = fopen("sample.txt", "r");

    if (fp == NULL) {

        printf("%s\n", "Error opening file!");

        exit(0);

    }

    fp2 = fopen("sample2.txt", "r");

    if (fp2 == NULL) {

        printf("%s\n", "Error opening file!");

        exit(1);

    }

    return 0;

}

## Mở file

### Mở file trong C bằng hàm fopen

fp = fopen(filepath, mode);

Trong đó:

* fp là con trỏ file dùng để gán kiểu cấu trúc FILE được trả về từ hàm nếu mở file thành công
* filepath là đường dẫn tới file cần mở. Đường dẫn này có thể là đường dẫn tương đối, hoặc là đường dẫn tuyệt đối.
* mode là chế độ mở file.

| **Mode** | **Xử lý** | **Chức năng** |
| --- | --- | --- |
| r | Mở để đọc | Chỉ cho phép đọc file Nếu file không tồn tại thì trả về NULL |
| w | Mở để ghi đè | Xoá nội dung cũ và ghi đè nội dung mới Nếu file không tồn tại thì tạo file mới |
| a | Mở để ghi chèn | Ghi chèn nội dung mới vào cuối file Nếu file không tồn tại thì tạo file mới |
| r+ | Mở để đọc và ghi đè | Cho phép cả đọc và ghi đè Nếu file không tồn tại thì trả về NULL |
| w+ | Mở để đọc và ghi đè | Cho phép cả đọc và ghi đè Nếu file không tồn tại thì tạo file mới |
| a+ | Mở để đọc và ghi chèn | Cho phép cả đọc và ghi chèn Nếu file không tồn tại thì tạo file mới |

VD

#include <stdio.h>

int main(void)

{

    //Khai báo con trỏ file

    FILE \* fp = NULL;

    //Mở file và gán file handle vào con trỏ file

    fp = fopen("sample.txt", "r");

    return 0;

}

### Mở file binary file

Khác với các tập tin văn bản mà chúng ta có thể đọc được các chữ cái, văn bản ghi trong đó, thì Binary File là một tập tin chứa nội dung là các chuỗi nhị phân như sau:  


Bằng cách thêm ký tự b vào đằng trước tên mode của hàm fopen thì chúng ta cũng có thể mở các binary file - tập tin nhị phân trong C.

Các mode dùng để mở **binary file** trong C sẽ là:

| **Mode** | **Xử lý** | **Chức năng** |
| --- | --- | --- |
| rb | Đọc | Chế độ đọc file binary |
| wb | Ghi đè | Chế độ ghi đè file binary |
| ab | Ghi chèn | Chế độ ghi chèn file binary |

### Mở file trong C bằng hàm fopen\_s

Hàm fopen\_s trong C là một phiên bản an toàn và bảo mật hơn của hàm fopen, và hàm này cũng giúp chúng ta mở file trong C.

Chúng ta sử dụng hàm fopen\_s trong C với cú pháp sau đây:

errno\_t err = fopen\_s(fp, filepath, mode);

Trong đó:

* errno\_t là kiểu lỗi trong C, và err là tên biến để gán giá trị lỗi nếu mở file thất bại
* fp là con trỏ file dùng để gán kiểu cấu trúc FILE được trả về từ hàm nếu mở file thành công
* filepath là đường dẫn tới file cần mở. Đường dẫn này có thể là đường dẫn tương đối, hoặc là đường dẫn tuyệt đối.
* mode là chế độ mở file. Các mode có thể dùng tương tự như với hàm fopen mà chúng ta đã học ở trên.

Hàm fopen\_s sẽ trả về một số tự nhiên biểu thị số hiệu của lỗi khi mở file. Nếu số này bằng 0 thì việc mở file thành công, và nếu số này khác 0 thì chúng ta có thể tìm ra nội dung lỗi bằng cách đối chiếu số hiệu của lỗi với bảng lỗi.

Trong trường hợp không cần xác định tới kiểu lỗi nếu có khi mở file, chúng ta cũng có thể lược bỏ đi việc gán errno\_t err mà dùng hàm trực tiếp với cú pháp sau đây:

fopen\_s(fp, filepath, mode);

Sau khi mở file thành công, thực thể được tạo ra từ **kiểu cấu trúc FILE** chứa các thông tin của file đã mở sẽ được gán vào con trỏ fp đã khai báo. Và chúng ta có thể thực hiện các thao tác với file thông qua con trỏ file này như bình thường.

VD

#include <stdio.h>

int main(void){

    //Mở file bằng hàm fopen\_s mà không kiểm tra lỗi

    FILE \* fp1 = NULL;

    fopen\_s(&fp1, "sample.txt", "r");

    fclose(fp1);

    //Mở file bằng hàm fopen\_s có kiểm tra lỗi

    FILE \* fp2 = NULL;

    errno\_t no2;

    no2 = fopen\_s(&fp2, "memo.txt", "w");

    if (no2 != 0) printf("%s\n", "Mo file thanh cong");

    fclose(fp2);

}

### Sự khác biệt giữa hàm fopen và fopen\_s

Hàm fopen\_s là một phiên bản an toàn và bảo mật hơn của hàm fopen. Do đó, hàm này cấm mở một file cùng lúc khi đang mở file đó ở chế độ ghi.

Bởi vậy, nếu chúng ta đã mở một file bằng hàm fopen\_s rồi thì chương trình tiếp sẽ kết thúc bất thường vào lần gọi thứ hai của hàm fopen\_s.

Sự khác biệt lớn nhất giữa giữa hàm fopen và fopen\_s cũng chính là điều này. Chúng ta có thể mở một file cùng lúc khi đang mở file đó ở chế độ ghi bằng hàm fopen, nhưng sẽ xảy ra lỗi nếu chúng ta làm việc này với hàm fopen\_s.

Ví dụ, do hàm fopen\_s được gọi 2 lần với cùng môt file, nên chương trình sẽ tự động kết thúc như sau:

VD1

#include <stdio.h>

int main(void)

{

    FILE \* fp1 = NULL;

    FILE \* fp2 = NULL;

    errno\_t no1;

    errno\_t no2;

    //Lần mở file đầu tiên với fopen\_s

    no1 = fopen\_s(&fp1, "memo.txt", "w");

    if (no1 != 0)

    {

        printf("ERR(%d): mo lan 1", no1);

    }

    //Lần mở cùng một file thứ 2 với fopen\_s

    no2 = fopen\_s(&fp2, "memo.txt", "w");

    if (no2 != 0)

    {

        printf("ERR(%d): mo lan 2", no2);

    }

    return 0;

}

RR(13): mo lan 2

### Xử lý lỗi khi mở file

Như đã phân tích trong bài Xử lý file trong c thì chúng ta cần phải xử lý lỗi khi mở một file thất bại trong C, để chương trình tự xử lý lỗi, và chạy chương trình một cách chính xác.

Và để xử lý lỗi khi mở file trong C, chúng ta hay sử dụng tới cách viết như sau đây:

//Mở file bằng hàm fopen, và trả về NULL nếu mở file thất bại.

fp = fopen(fname, "r");

if(fp == NULL) {

    printf("%s file not open!\n", fname);

    exit(0);

}

Lưu ý nếu bạn mở file bên trong hàm main(), thì bằng cách dùng lệnh return như sau thì chúng ta cũng có thể xử lý lỗi khi mở file trong C:

//Mở file bằng hàm fopen, và trả về NULL nếu mở file thất bại.

fp = fopen(fname, "r");

if(fp == NULL) {

    printf("%s file not open!\n", fname);

    return -1;

}

### Tạo file mới

Bằng cách sử dụng một trong 2 hàm mở file trên, chúng ta có thể tạo một file mới và mở nó ra để xử lý. Ví dụ chúng ta sẽ tạo một file mới có tên *sample.txt* bằng hàm fopen() với mode **w** như sau:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL; //Khai báo con trỏ file

    fp = fopen("sample.txt", "w"); // Mở file và gán file handle vào con trỏ file

    return 0;

}

Bạn cũng có thể sử dụng mode a+ để tạo file mới trong C:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL; //Khai báo con trỏ file

    fp = fopen("sample.txt", "a+"); // Mở file và gán file handle vào con trỏ file

    return 0;

}

Sau khi compile và chạy chương trình, một file trống memo.txt với kích thước 0 KB sẽ được tạo ra như dưới đây:



**Đóng file trong C bằng hàm fclose**

int fclose(FILE \* fp);

VD

#include <stdio.h>

int main(void){

    //Mở file

    FILE \* fp = NULL;

    fp = fopen("memo.txt", "w");

    //Đóng file

    fclose(fp);

    return 0;

}

## fgetc

### Đọc từng ký tự trong file bằng hàm fgetc

Hàm fgetc trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng **đọc từng ký tự trong file** chỉ định. Tên hàm fgetc được viết tắt bởi cụm từ *file*, *get* và *character*, được dịch theo tiếng Việt chính xác là *hàm đọc từng ký tự trong file*.

int fgetc(FILE \* fp);

Hàm fgetc sẽ trả về mã ASCII của 1 ký tự được đọc ra từ file. Trong trường hợp vị trí đọc ký tự đã là cuối file, hoặc là việc đọc file thất bại thì giá trị EOF sẽ được trả về. Bằng việc in mã ASCII này sang dạng char, chúng ta có thể lấy xuất được ký tự đã đọc.

VD02

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    //Mở file bằng hàm fopen

    fp= fopen("sample.txt", "r");

    printf("%c\n", fgetc(fp));

    printf("%c\n", fgetc(fp));

}

h

e

Bạn có thể thấy chúng ta chỉ có thể đọc từng ký tự từ file trong mỗi lần chạy hàm fgetc mà thôi.

Để có thể đọc tất cả các ký tự từ trong file, chúng ta sẽ cần tạo ra một vòng lặp để đọc từng ký tự từ đầu file cho tới cuối file, cho tới khi kết quả trả về là giá trị **EOF** như sau:

VD3

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    //Mở file bằn hàm fopen

    fp= fopen("sample.txt", "r");

    char c;

    //Đọc từng ký tự từ file cho tới khi gặp EOF

    while ((c = fgetc(fp)) != EOF)

    {

        //Xuất từng ký tự ra màn hình

        printf("%c", c);

    }

    fclose(fp);

    return 0;

}

hello

how are you

Lưu ý ở đây, các ký tự xuống dòng cũng được đọc từ file và do đó kết quả xuất ra màn hình cũng được xuống dòng như trên.

## fgets

### Đọc từng dòng file trong C bằng hàm fgets

Chúng ta sử dụng hàm fgets trong C với cú pháp sau đây:

char \* fgets(char \* buf, int size, FILE \* fp);

Trong đó:

* fp là con trỏ của file cần đọc, được tạo ra từ việc mở file
* buf là con trỏ tới nơi lưu trữ chuỗi đã đọc từ dòng trong file. Thông thường chúng ta chỉ định buf bằng một mảng.
* size là kích thước (số ký tự) lớn nhất có thể đọc từ dòng trong file.

Hàm fgets sẽ trả về con trỏ lưu địa chỉ trên bộ nhớ của chuỗi được đọc từ dòng trong file. Trong trường hợp vị trí đọc ký tự đã là cuối file, hoặc là việc đọc file thất bại thì con trỏ NULL sẽ được trả về.

Lại nữa, hàm fgets sẽ đọc một dòng từ file, và dòng đó được tính từ đầu dòng cho tới khi gặp ký tự xuống dòng \n.

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    char arr[128];

    //Mở file bằn hàm fopen

    fp= fopen("nums.txt", "r");

    //Đọc dòng 1

    fgets(arr, 128, fp);

    printf("%s", arr);

    //Đọc dòng 2

    fgets(arr, 128, fp);

    printf("%s", arr);

}

1234567

89abc

Bạn có thể thấy chúng ta chỉ có thể đọc từng dòng từ file trong mỗi lần chạy hàm fgetc mà thôi.

Để có thể đọc tất cả các dòng từ trong file, chúng ta sẽ cần tạo ra một vòng lặp để đọc từng dòng từ đầu file cho tới cuối file, cho tới khi kết quả trả về là giá trị NULL như sau:

VD5

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    char arr[128];

    //Mở file bằn hàm fopen

    fp= fopen("nums.txt", "r");

    //Đọc từng dòng từ file cho tới khi gặp NULL

    while (fgets(arr, 128, fp) != NULL)

    {

        //Xuất từng dòng ra màn hình

        printf("%s", arr);

    }

    fclose(fp);

    return 0;

}

1234567

89abc

def

### Đọc từng dòng file khi đối số size nhỏ hơn số ký tự

Đối số size trong hàm fgets giúp chúng ta chỉ định số ký tự lớn nhất có thể được đọc từ một dòng trong file. Thông thường chúng ta chỉ định giá trị của size bằng với số phần tử của mảng chuẩn bị để lưu nội dung đọc từ dòng đó.

Lưu ý trong trường hợp giá trị của size nhỏ hơn số ký tự thực có trong dòng, thì chỉ có **size -1** ký tự thực được đọc ra từ dòng đó mà thôi. Chỗ trống của ký tự còn lại sẽ được tự động lấp chỗ bằng một ký tự kết thúc chuỗi \0.

Ví dụ, chúng ta sẽ đọc lại file *nums.txt* ở trên, nhưng với chỉ định số phần tử lớn nhất có thể đọc lại nhỏ hơn số ký tự thực trong dòng như sau:

VD

Nội dung file nums.txt

1234567

89abc

def

Code

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    char arr[128];

    //Mở file bằn hàm fopen

    fp= fopen("nums.txt", "r");

    //Đọc dòng 1

    fgets(arr, 3, fp);

    printf("%s\n", arr);

    //Đọc dòng 2

    fgets(arr, 4, fp);

    printf("%s", arr);

}

12

345

Ở lần đọc thứ 2, hàm fgets cũng sẽ bắt đầu đọc từ cùng một dòng ban đầu do dòng này chưa được đọc hết ký tự. Và vị trí bắt đầu được đọc chính là từ ký tự kết thúc chuỗi \0 mới được điền ở lần đọc đầu tiên.

## fscanf

### Đọc từng dòng file theo định dạng chỉ định bằng hàm fscanf

Fscanf giúp đọc từng dòng trong file nhưng có chọn lọc. ví dụ như khi chúng ta chỉ muốn đọc chuỗi từ file trong c, hoặc là khi chỉ muốn đọc số từ file trong c.

Hàm này đặc biệt hữu dụng khi cần đọc các file được viết theo định dạng cố định. Ví dụ như khi chúng ta cần đọc file csv trong c với các ô trong file được phân cách bởi dấu phẩy, hoặc là bằng dấu cách chẳng hạn.

Hàm fscanf không đọc toàn bộ nội dung của dòng, mà sẽ phân tách nội dung dòng đó theo từng định dạng chỉ định. Các dữ liệu được truy xuất từ dòng theo định dạng chỉ định cũng sẽ được lưu trữ với định dạng tương ứng trong chương trình.

Chúng ta sử dụng hàm fscanf trong C với cú pháp sau đây:

int fscanf(FILE \* fp, "fo1 fo2 fo3 ... ", add1, add2, add3 ...);

Trong đó

* fp là con trỏ của file cần đọc, được tạo ra từ việc mở file ở phần trên.
* Các cặp fo và add tương ứng là định dạng (format) của dữ liệu cần đọc từ dòng, và địa chỉ của tên biến (con trỏ biến) dùng để lưu dữ liệu đó trong bộ nhớ.

Hàm fscanf sẽ trả về một số thuộc kiểu int, chính là số mục có thể đọc được từ dòng. Trong trường hợp dòng được đọc đã là dòng cuối file, hoặc là việc đọc file thất bại thì giá trị EOF sẽ được trả về.

Định dạng (format) của dữ liệu cần đọc cũng như kiểu của biến lưu dữ liệu được tóm tắt trong bảng dưới đây:

| **Định dạng chuyển đổi** | **Kiểu biến** | **Chi tiết** |
| --- | --- | --- |
| %hhd | char unsigned char | Chuyển về dạng thập phân và lưu trữ trong biến 1 byte |
| %hd | short unsigned short | Chuyển về dạng thập phân và lưu trữ trong biến 2 byte |
| %d | int unsigned int | Chuyển về dạng thập phân và lưu trữ trong biến kiểu int |
| %ld | long unsigned long | Chuyển về dạng thập phân và lưu trữ trong biến 4 byte |
| %hhx | char unsigned char | Chuyển về hệ thập lục phân và lưu trữ trong biến 1 byte |
| %hx | short unsigned short | Chuyển về hệ thập lục phân và lưu trữ trong biến 2 byte |
| %x | int unsigned int | Chuyển về hệ thập lục phân và lưu trữ trong biến kiểu int |
| %lx | long unsigned long | Chuyển về hệ thập lục phân và lưu trữ trong biến 4 byte |
| %f | float | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu float |
| %lf | double | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu double |
| %c | char | Chuyển về 1 ký tự và lưu trữ trong biến kiểu char |
| %s | char \* | Chuyển về chuỗi ký tự và lưu trữ trong biến kiểu mảng char |
| %p | void \* | Chuyển về địa chỉ và lưu trữ trong con trỏ |

### Đọc số từ file

Ví dụ, chúng ta có file user.txt với nội dung sau đây:

Kiyoshi 32 172.5cm A

Honda 24 185.3cm O

Suzuki 63 153.8cm B

Giả sử chúng ta chỉ muốn đọc dữ liệu số bao gồm cột *tuổi* và *chiều cao* trong từng dòng file trên, khi đó chúng ta dùng hàm fscanf như sau:

#include <stdio.h>

int main(void){

    FILE \* fp = NULL;

    char    name[32] = { 0 };    // Tên

    int        age = 0;            // Tuổi

    double    height = 0;            // Chiều cao

    char    blood = 0;            // Nhóm máu

    //Mở file bằng hàm fopen\_s

    fopen\_s(&fp, "user.txt", "r");

    //Đọc từng dòng trong file cho tới khi gặp EOF

    while (fscanf(fp, "%s %d %lfcm %c", name, &age, &height, &blood) != EOF)

    {

        //Xuất các dữ liệu số cần đọc

        printf("%d %3.2lf \n", age, height);

    }

    return 0;

}

32 172.50

24 185.30

63 153.80

Lưu ý là khi đọc từng dòng trong file bằng hàm fscanf thì chúng ta cần đọc tất cả các mục có thể đọc ra, tuy nhiên khi cần xuất dữ liệu số thì chúng ta chỉ cần chỉ định các dữ liệu đó mà thôi.

### Đọc chuỗi từ file

Tương tự như trên thì bằng cách chỉ định định dạng dữ liệu cần đọc dưới dạng số trong hàm fscanf, chúng ta có thể tiến hành đọc chuỗi từ file trong c.

Ví dụ, chúng ta cũng đọc file user.txt với nội dung sau đây:

Kiyoshi 32 172.5cm A

Honda 24 185.3cm O

Suzuki 63 153.8cm B

Giả sử chúng ta chỉ muốn đọc dữ liệu chuỗi bao gồm cột tên và nhóm máu trong từng dòng file trên, khi đó chúng ta dùng hàm fscanf như sau:

#include <stdio.h>

int main(void){

    FILE \* fp = NULL;

    char    name[32] = { 0 };    // Tên

    int        age = 0;            // Tuổi

    double    height = 0;            // Chiều cao

    char    blood = 0;            // Nhóm máu

    //Mở file bằng hàm fopen\_s

    fopen\_s(&fp, "user.txt", "r");

    //Đọc từng dòng trong file cho tới khi gặp EOF

    while (fscanf(fp, "%s %d %lfcm %c", name, &age, &height, &blood) != EOF)

    {

        //Xuất các dữ liệu số cần đọc

        printf("%s %c\n", name, blood);

    }

    return 0;

}

Kiyoshi A

Honda O

Suzuki B

Lưu ý là khi đọc từng dòng trong file bằng hàm fscanf thì chúng ta cần đọc tất cả các mục có thể đọc ra, tuy nhiên khi cần xuất dữ liệu chuỗi thì chúng ta chỉ cần chỉ định các dữ liệu đó mà thôi.

## Sscanf

### Đọc từng dòng file theo định dạng chỉ định bằng hàm sscanf

Hàm sscanf trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng truy xuất thông tin theo định dạng chỉ định từ một chuỗi ký tự. Bằng cách ứng dụng hàm sscanf, chúng ta có thể lấy ra các thông tin cần thiết từ chuỗi theo một định dạng chỉ định nào đó.

Cú pháp sử dụng hàm sscanf trong C như sau:

int sscanf (buff, "fo1 fo2 fo3 ...", add1, add2, add3 ...);

Trong đó

* buff là con trỏ tới chuỗi ký tự cần phân tích để lấy ra thông tin theo định dạng
* Các cặp fo và add tương ứng là định dạng (format) của dữ liệu cần đọc từ chuỗi ký tự, và địa chỉ của tên biến dùng để lưu thông tin được tách ra từ chuỗi trên bộ nhớ.

Cách sử dụng của nó rất giống với hàm fscanf ở trên, ngoại trừ việc nó sẽ xử lý nội dung dòng được đọc bởi hàm fgets mà thôi.

Ví dụ cụ thể, chúng ta có file test.txt với nội dung sau đây:

test01 1.0 1.1 1.2 1.3 1.4

test02 2.0 2.1 2.2 2.3 2.4

test03 3.0 3.1 3.2 3.3 3.4

Bằng việc sử dụng hàm fgets để đọc từng dòng trong file này, sau đó dùng hàm sscanf để lấy thông tin cần thiết theo định dạng từ dòng, mà chúng ta có thể đọc từng dòng trong file với định dạng chỉ định như sau:

#include <stdio.h>

#define N 256 //Chỉ định số ký tự lớn nhất có thể đọc từ một dòng

int main(void) {

    FILE \*fp;

    char fname[] = "test.txt";

    char line[N];

    char str[16];

    float f1, f2, f3, f4, f5;

    //Mở file bằng hàm fopen, và trả về NULL nếu mở file thất bại.

    fp = fopen(fname, "r");

    if(fp == NULL) {

        printf("%s file not open!\n", fname);

        return -1;

    }

    //Đọc từng dòng trong file bằng hàm fgets

    while(fgets(line, N, fp) != NULL) {

        //Truy xuất thông tin cần thiết từ nội dung đọc được bằng hàm sscanf

        sscanf(line, "%s %f %f %f %f %f", str, &f1, &f2, &f3, &f4, &f5);

        printf("%s %.1f %.1f %.1f %.1f %.1f\n", str, f1, f2, f3, f4, f5);

    }

    fclose(fp); //Đóng file

    return 0;

}

test01 1.0 1.1 1.2 1.3 1.4

test02 2.0 2.1 2.2 2.3 2.4

test03 3.0 3.1 3.2 3.3 3.4

## Đọc file csv

### Mở file CSV có các cột phân tách bởi dấu cách

Giả sử chúng ta có file test.csv với nội dung sau đây:

test01 1.0 1.1 1.2 1.3 1.4

test02 2.0 2.1 2.2 2.3 2.4

test03 3.0 3.1 3.2 3.3 3.4

Chúng ta sẽ mở file này và đọc từng dòng trong file theo định dạng như sau:

#include <stdio.h>

int main(void) {

    FILE \*fp;

    char fname[] = "test.csv";

    char str[16];

    float f1, f2, f3, f4, f5;

    fp = fopen(fname, "r"); //Mở file bằng hàm fopen

    //Xử lý lỗi khi mở file thất bại

    if(fp == NULL) {

        printf("%s file not open!\n", fname);

        return -1;

    }

    //Đọc từng dòng theo định dạng bằng hàm fscan

    while(fscanf(fp, "%s %f %f %f %f %f", str, &f1, &f2, &f3, &f4, &f5) != EOF) {

        printf("%s %.1f %.1f %.1f %.1f %.1f\n", str, f1, f2, f3, f4, f5);

    }

    fclose(fp); //Đóng file

    return 0;

}

test01 1.0 1.1 1.2 1.3 1.4

test02 2.0 2.1 2.2 2.3 2.4

test03 3.0 3.1 3.2 3.3 3.4

### Mờ file CSV có các cột phân tách bởi dấu phẩy

Giả sử chúng ta có file test.csv với nội dung sau đây:

test01,1.0,1.1,1.2,1.3,1.4

test02,2.0,2.1,2.2,2.3,2.4

test03,3.0,3.1,3.2,3.3,3.4

Một cách tương tự thì chúng ta cũng mở file này và đọc từng dòng trong file theo định dạng như sau:

#include <stdio.h>

int main(void) {

    FILE \*fp;

    char fname[] = "test.csv";

    char str[16];

    float f1, f2, f3, f4, f5;

    fp = fopen(fname, "r"); //Mở file bằng hàm fopen

    //Xử lý lỗi khi mở file thất bại

    if(fp == NULL) {

        printf("%s file not open!\n", fname);

        return -1;

    }

    //Đọc từng dòng theo định dạng bằng hàm fscan

    while(fscanf(fp, "%[^,],%f,%f,%f,%f,%f", str, &f1, &f2, &f3, &f4, &f5) != EOF) {

        printf("%s,%.1f,%.1f,%.1f,%.1f,%.1f", str, f1, f2, f3, f4, f5);

    }

    fclose(fp); //Đóng file

    return 0;

}

test01,1.0,1.1,1.2,1.3,1.4

test02,2.0,2.1,2.2,2.3,2.4

test03,3.0,3.1,3.2,3.3,3.4

### Đọc mảng từ file

Đối với các file có định dạng cố định như là CSV, thì khi chúng ta cũng có thể đọc mảng từ file trong C.

Ví dụ, chúng ta có file test.csv có nội dung sau đây:

test01,1.0,1.1,1.2,1.3,1.4

test02,2.0,2.1,2.2,2.3,2.4

test03,3.0,3.1,3.2,3.3,3.4

Có thể thấy với file CSV này thì các ô trong hàng được cách nhau bởi dấu phẩy, trong đó ô đầu tiên ở dạng chuỗi ký tự, và các ô sau thì ở dạng số thực.

Do các dòng trong file đều có chung cấu trúc như vậy, nên sau khi đọc từng dòng của file bằng hàm fscanf, hoặc là tổ hợp hàm fgets và hàm sscanf mà Kiyoshi đã giới thiệu ở trên, chúng ta có thể lưu chuỗi ký tự đọc được từ ô đầu tiên vào một mảng chuỗi, và các số đọc được từ các ô còn lại thì lưu vào trong mảng số khi đọc từng dòng trong file. Đây là cách đọc mảng từ file trong C.

Và với từng dòng như vậy, do tồn tại 2 mảng với 2 kiểu dữ liệu khác nhau, nên chúng ta sẽ nhóm hai loại dữ liệu khác nhau này trong một tập hợp cho dễ quản lý thông qua kiểu cấu trúc trong C.

Đây cũng chính là cách Đọc cấu trúc từ file trong C. Nói một cách dễ hiểu thì chúng ta sẽ chỉnh lý các dữ liệu đọc được từ file và lưu chúng vào các mảng và kiểu dữ liệu để dễ dàng xử lý chúng trong chương trình.

Tất nhiên sau khi đã lưu chúng vào mảng hoặc cấu trúc rồi, chúng ta hoàn toàn có thể sử dụng các phương pháp xử lý mảng hoặc phương pháp xử lý cấu trúc để xử lý dữ liệu đọc được từ chuỗi. Bạn cũng có thể tham khảo các phương pháp này tại hai chuyên đề là Mảng trong C và Kiểu cấu trúc trong C.

Và chúng ta sẽ đọc mảng từ file với chương trình như sau:

#include <stdio.h>

#define N 256 //Định nghĩa N là số ký tự lớn nhất có thể đọc từ một dòng

#define ROW 3 //Định nghĩa số dòng sẽ đọc trong file

// Khai báo cấu trúc mảng để lưu chuỗi và số trích xuất từ dòng

typedef struct str {

    char str[16];

    float f\_data[5];

} data;

int main(void) {

    FILE \*fp; //

    char fname[] = "test.csv";

    char line[N];

    char str[16];

    float f1, f2, f3, f4, f5;

    int i = 0;

    //Khởi tạo thực thể từ kiểu cấu trúc data

    data data[ROW];

    fp = fopen(fname, "r"); //Mở file bằng hàm fopen, trả về NULL khi thất bại

    if(fp == NULL) {

        printf("%s file not open!\n", fname);

        return -1;

    }

    //Đọc từng dòng trong file bằng hàm fgets

    while(fgets(line, N, fp) != NULL) {

        //Trích xuất thông tin từ nội dung file bằng hàm sscanf

        sscanf(line, "%[^,],%f,%f,%f,%f,%f", str, &f1, &f2, &f3, &f4, &f5);

        //Gán các nội dung trích xuất được lần lượt vào trong thực thể của cấu trúc

        for(int j = 0; j  < sizeof(data[i].str) / sizeof(data[i].str[0]); j++) {

            data[i].str[j] = str[j];

        }

        float tmp[] = {f1, f2, f3, f4, f5};

        for(int j = 0; j  < sizeof(data[i].f\_data) / sizeof(data[i].f\_data[0]); j++) {

            data[i].f\_data[j] = tmp[j];

        }

        //Xuất giá trị các thành viên trong thực thể của cấu trúc

        printf("%s,%.1f,%.1f,%.1f,%.1f,%.1f\n", data[i].str, data[i].f\_data[0], data[i].f\_data[1], data[i].f\_data[2], data[i].f\_data[3], data[i].f\_data[4]);

        i++;

    }

    fclose(fp); //Đóng file

    return 0;

}

test01,1.0,1.1,1.2,1.3,1.4

test02,2.0,2.1,2.2,2.3,2.4

test03,3.0,3.1,3.2,3.3,3.4

Lưu ý trong định dạng dữ liệu sử dụng trong hàm sscanf, chúng ta đã dùng tới biểu thức chính quy với cách viết %[^,], có tác dụng lấy các dữ liệu chuỗi ngoại trừ dấu phẩy. Lý do là dấu phẩy được công nhận là một phần của chuỗi, do đó, chúng ta cần dùng biểu thức chính quy để loại bỏ dấu phẩy này từ trong kết quả đọc chuỗi.

## Ghi file

### Ghi ký tự vào file bằng hàm fputc

Hàm fputc trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng ghi từng ký tự vào file chỉ định. Tên hàm fputc được viết tắt bởi cụm từ file, put và character, được dịch theo tiếng Việt chính xác là hàm ghi từng ký tự vào file.

Chúng ta sử dụng hàm fputc trong C với cú pháp sau đây:

int fputc(int char, FILE \* fp);

Trong đó:

* fp là con trỏ của file cần ghi, được tạo ra từ việc mở file ở phần trên.
* char là ký tự cần ghi vào file
* Hàm fputc sẽ trả về ký tự đã được ghi vào file. Trong trường việc ghi file thất bại thì giá trị EOF sẽ được trả về.

Ví dụ cụ thể, chúng ta sẽ ghi dòng text Hello vào file trống sample.txt. Chúng ta sẽ dùng hàm fopen để mở file, sau đó ghi từng ký tự có trong dòng nội dung này vào file bằng hàm fputc như sau:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    fopen\_s(&fp, "sample.txt", "w");

    fputc('H', fp);

    fputc('e', fp);

    fputc('l', fp);

    fputc('l', fp);

    fputc('o', fp);

    fputc('\n', fp);

    fclose(fp);

    return 0;

}

Hello

**Ghi chuỗi vào file trong C bằng hàm fputs**

Hàm fputs trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng ghi từng chuỗi vào file chỉ định.

Chúng ta sử dụng hàm fputs trong C với cú pháp sau đây:

int fputs(const char \* str, FILE \* fp);

Trong đó:

* fp là con trỏ của file cần ghi, được tạo ra từ việc mở file
* str là con trỏ tới nơi lưu trữ chuỗi đã ghi từ dòng vào file. Thông thường chúng ta chỉ định str bằng một chuỗi.

Hàm fputs sẽ trả về 0 nếu việc ghi chuỗi vào file thành công. Trong trường hợp việc ghi file thất bại thì EOF sẽ được trả về.

Lưu ý là hàm fputs không tự động ghi ký tự xuống dòng \n sau khi đã ghi chuỗi, do đó khi muốn xuống dòng khi ghi chuỗi thì chúng ta phải tự điền ký tự \n vào vị trí cần xuống dòng trong chuỗi.

Ví dụ cụ thể, chúng ta sẽ ghi nội dung sau vào file trống Hello.txt.

Hello World!

I am Kiyoshi

Chúng ta sẽ dùng hàm fopen để mở file, sau đó ghi các chuỗi này vào file bằng hàm fputs như sau:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    fopen\_s(&fp, "Hello.txt", "w");

    fputs("Hello World!\nI am Kiyoshi", fp);

    fclose(fp);

    return 0;

}

Hello World!

I am Kiyoshi

### Ghi dữ liệu vào file trong C theo định dạng bằng hàm fprintf

int fprintf(fp, “fo1 fo2 fo3 “, var1, var2, var3 );

Trong đó

* fp là con trỏ của file cần ghi, được tạo ra từ việc mở file ở phần trên.
* Các cặp fo và var tương ứng là định dạng (format) ghi vào file của dữ liệu, và tên biến chứa dữ liệu đó.

Cơ chế hoạt động của hàm fprintf là lấy dữ liệu từ biến var, định dạng nó theo định dạng ghi trong fo để tạo ra một chuỗi. Sau đó thì ghi chuỗi vừa tạo này vào file mà con trỏ fp chỉ đến.

Hàm fprintf sẽ trả về một số thuộc kiểu int, chính là số ký tự có trong chuỗi được tạo. Trong trường hợp việc ghi file thất bại thì giá trị EOF sẽ được trả về.

Định dạng (format) của dữ liệu cần ghi cũng như kiểu của biến lưu dữ liệu được tóm tắt trong bảng dưới đây:

| **Định dạng chuyển đổi** | **Kiểu biến** | **Chi tiết** |
| --- | --- | --- |
| %hhd | char unsigned char | Chuyển về dạng thập phân và lưu trữ trong biến 1 byte |
| %hd | short unsigned short | Chuyển về dạng thập phân và lưu trữ trong biến 2 byte |
| %d | int unsigned int | Chuyển về dạng thập phân và lưu trữ trong biến kiểu int |
| %ld | long unsigned long | Chuyển về dạng thập phân và lưu trữ trong biến 4 byte |
| %hhx | char unsigned char | Chuyển về hệ thập lục phân và lưu trữ trong biến 1 byte |
| %hx | short unsigned short | Chuyển về hệ thập lục phân và lưu trữ trong biến 2 byte |
| %x | int unsigned int | Chuyển về hệ thập lục phân và lưu trữ trong biến kiểu int |
| %lx | long unsigned long | Chuyển về hệ thập lục phân và lưu trữ trong biến 4 byte |
| %f | float | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu float |
| %lf | double | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu double |
| %c | char | Chuyển về 1 ký tự và lưu trữ trong biến kiểu char |
| %s | char \* | Chuyển về chuỗi ký tự và lưu trữ trong biến kiểu mảng char |
| %p | void \* | Chuyển về địa chỉ và lưu trữ trong con trỏ |

Giả sử chúng ta có các kiểu thông tin như [chuỗi], [ký tự] và [số] được lưu vào các biến tương ứng. Bằng cách sử dụng hàm fprintf, chúng ta có thể định dạng các thông tin này và ghi vào file như chương trình sau đây:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    char name[] = "Kiyoshi";

    char sex = 'M';

    int  old = 30;

    fopen\_s(&fp, "user.txt", "w");

    // Ghi dữ liệu theo định dạng chỉ định vào file

    fprintf(fp, "%s %c %d\n", name, sex, old);

    fprintf(fp, "%s,%c,%d\n", name, sex, old);

    fprintf(fp, "name: %s, sex: %c, olde: %d\n", name, sex, old);

    fclose(fp);

    return 0;

}

Kiyoshi M 30

Kiyoshi,M,30

name: Kiyoshi, sex: M, old: 30

### Ghi mảng vào file

Ứng dụng hàm fprintf, chúng ta có thể thực hiện ghi mảng vào file trong c.

Do các phần tử trong mảng C đều có cùng kiểu dữ liệu, nên chúng ta có thể chỉ định cùng kiểu định dạng để ghi các phần tử từ mảng vào file bằng hàm hàm fprintf. Ví dụ cụ thể:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    int num[] = {10, 20, 44, 60, 82};

    fopen\_s(&fp, "num.txt", "w");

    // Ghi dữ liệu theo định dạng chỉ định vào file

    fprintf(fp, "%d %d %d %d %d", num[0], num[1], num[2], num[3], num[4]);

    fclose(fp);

    return 0;

}

10 20 44 60 82

Một cách tương tự thì chúng ta cũng có thể ghi các mảng 2 chiều vào file trong C. Khi đó mỗi mảng con 1 chiều trong mảng 2 chiều sẽ được coi như một dòng để ghi vào file, và chúng ta cần phải thêm ký tự xuống dòng \n khi ghi từng mảng dòng vào file như sau:

#include <stdio.h>

int main(void)

{

    FILE \* fp = NULL;

    int bangdiem[][3] = {

    {7, 9, 8} ,

    {8, 6, 7} ,

    {5, 7, 6} ,

    {4, 9, 5} ,

    {5, 8, 7} ,

    {6, 9, 3}

} ;

    //Lấy độ dài của mảng 2 chiều

    int y = sizeof(bangdiem) / sizeof(bangdiem[0]);

    int x = sizeof(bangdiem[0]) / sizeof(bangdiem[0][0]);

    fopen\_s(&fp, "bangdiem.txt", "w");

    // Ghi từng mảng 1 chiều trong mảng 2 chiều vào file

    for (int i = 0; i < y; i++){

        fprintf(fp, "Toan: %d Van: %d Anh: %d\n", bangdiem[i][0], bangdiem[i][1], bangdiem[i][2]);

    }

    fclose(fp);

    return 0;

}

Toan: 7 Van: 9 Anh: 8

Toan: 8 Van: 6 Anh: 7

Toan: 5 Van: 7 Anh: 6

Toan: 4 Van: 9 Anh: 5

Toan: 5 Van: 8 Anh: 7

Toan: 6 Van: 9 Anh: 3

### Ghi cấu trúc vào file

Ứng dụng hàm fprintf, chúng ta có thể thực hiện ghi cấu trúc vào file trong c.

Do các thành viên trong cấu trúc C có thể có kiểu dữ liệu khác nhau, nên chúng ta cần chỉ định kiểu định dạng phù hợp cho từng thành viên để ghi giá trị của chúng từ mảng vào file bằng hàm hàm fprintf.

Ví dụ chúng ta ghi dữ liệu từ mảng cấu trúc vào file trong C bằng hàm fprintf như sau:

#include <stdio.h>

#define PERSON\_NUM 5

//Khai báo kiểu struct

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

int main(void)

{

    //Khởi tạo thực thể struct là một mảng struct

    person\_t p[PERSON\_NUM] = {{"Bob",      'M', 19, 165.4, 72.5},

                              {"Alice",    'F', 19, 161.7, 44.2},

                              {"Tom",      'M', 20, 175.2, 66.3},

                              {"Stefany",  'F', 18, 159.3, 48.5},

                              {"Leonardo", 'M', 19, 172.8, 67.2}};

    //Tạo con trỏ file

    FILE \* fp = NULL;

    //Mở file

    fopen\_s(&fp, "people.txt", "w");

    //Ghi header vào file

    fprintf(fp, "%s %s %s %s %s\n", "name", "sex", "age", "height", "weight");

    // Ghi từng thực thể của cấu trúc như một dòng vào file

    for (short i = 0; i < PERSON\_NUM; i++) {

           fprintf(fp, "%s %c %d %.2f %.2f \n",p[i].name, p[i].sex,p[i].age,p[i].height,p[i].weight);

    }

    fclose(fp);

    return 0;

}

name sex age height weight

Bob M 19 165.40 72.50

Alice F 19 161.70 44.20

Tom M 20 175.20 66.30

Stefany F 18 159.30 48.50

Leonardo M 19 172.80 67.20

### Ghi file CSV

Ở phần trên, chúng ta đã tiến hành ghi nội dung các kiểu dữ liệu phức tạp như mảng hay cấu trúc vào file dưới dạng file txt rồi. Tuy nhiên chúng ta cũng có thể ghi dữ liệu trên vào file CSV trong C, đơn giản bằng cách thay đổi đuôi file từ .txt sang .csv là xong.

Ví dụ cụ thể, chúng ta ghi dữ liệu từ mảng cấu trúc vào file CSV trong C bằng hàm fprintf như sau. Lưu ý là chúng ta cần viết định dạng các dữ liệu cách nhau bởi dấu phẩy để có thể ghi từng dữ liệu vào từng ô trong file CSV.

#include <stdio.h>

#define PERSON\_NUM 5

typedef struct {

    char name[20];

    char sex;

    int age;

    double height;

    double weight;

} person\_t;

int main(void)

{

    person\_t p[PERSON\_NUM] = {{"Bob",      'M', 19, 165.4, 72.5},

                              {"Alice",    'F', 19, 161.7, 44.2},

                              {"Tom",      'M', 20, 175.2, 66.3},

                              {"Stefany",  'F', 18, 159.3, 48.5},

                              {"Leonardo", 'M', 19, 172.8, 67.2}};

    FILE \* fp = NULL;

    int num[] = {10, 20, 44, 60, 82};

    fopen\_s(&fp, "people.csv", "w");

    // Ghi từng mảng 1 chiều trong mảng 2 chiều vào file

   fprintf(fp, "%s,%s,%s,%s,%s\n", "name", "sex", "age", "height", "weight");

   for (short i = 0; i < PERSON\_NUM; i++) {

           fprintf(fp, "%s,%c,%d,%.2f,%.2f\n",p[i].name, p[i].sex,p[i].age,p[i].height,p[i].weight);

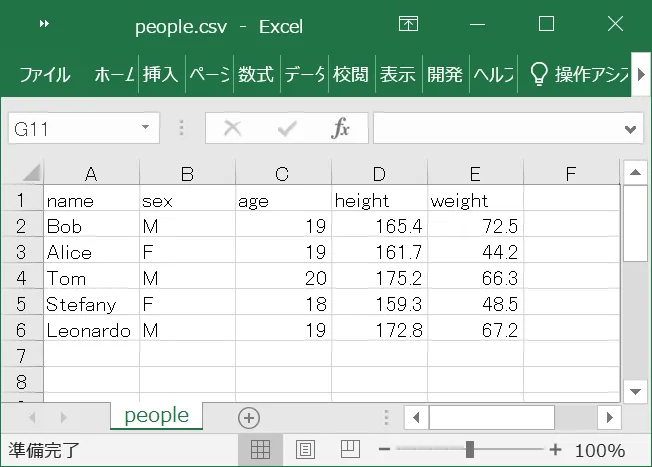
    }

    fclose(fp);

    return 0;

}

Kết quả, nội dung mảng cấu trúc sẽ được ghi vào file people.csv như sau:



# 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

# Hàm (function)

Khái niệm:

Hàm trong C là một tập hợp các xử lý nhằm thực hiện một chức năng cụ thể nào đó trong chương trình. Hàm cho phép bạn kết hợp các xử lý khác nhau thành một và đặt tên cho nó. Sau khi tạo và đặt tên cho một hàm, chúng ta chỉ cần gọi tên hàm ra mỗi khi cần sử dụng đến nó trong chương trình.

Tham khảo:

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/ham-trong-c/ham-trong-c/>

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/ham-trong-c/khai-bao-va-goi-ham-trong-c/>

Khái báo:

kiểu-trả-về tên-hàm ( kiểu-1 tham-số-1 , kiểu-2 tham-số-2 , ...){

Câu lệnh 1 trong hàm;

Câu lệnh 2 trong hàm;

...

return giá-trị-trả-về ;

}

Gọi hàm:

tên-hàm( đối-số-1 , đối-số-2 ,...);

## memset()

Cú pháp của hàm như sau:

void \* memset(void \* ptr, int value, size\_t num);

Tác dụng của hàm là set num byte giá trị của vùng nhớ tính từ vị trí ptr trỏ tới bằng giá trị value

Ví dụ:

#include <stdio.h>

#include <string.h>

int main()

{

    char str[30] = "We are Stdio's Writers";

    printf("%s\n", str);

    memset(str + 3, '@', 10); // set 10 byte từ vị trí thứ 4 của str thành @

    printf("%s\n", str);

    return 0;

}

Kết quả in ra màn hình sẽ là:

We are Stdio's Writers

We @@@@@@@@@@s Writers

Tuy memset() có công dụng khá hay, nhưng ít khi lập trình viên sử dụng đến, trừ trường hợp muốn set toàn bộ vùng nhớ thành 1 giá trị gì đó, thường là giá trị 0.

## memcpy()

Thuộc thư viện <string.h>

void \* memcpy(void \* destination, const void \* source, size\_t num);

Tác dụng của hàm là sao chép num byte từ vị trí mà source trỏ tới đến vị trí mà destination trỏ tới

VD

#include <stdio.h>

#include <string.h>

struct {

  char name[40];

  int age;

} person, person\_copy;

int main ()

{

  char myname[] = "Nguyen Hoang Nam";

  /\* su dung ham memcpy de sao chep chuoi: \*/

  memcpy ( person.name, myname, strlen(myname)+1 );

  person.age = 46;

  /\* su dung ham memcpy de sao chep struct: \*/

  memcpy ( &person\_copy, &person, sizeof(person) );

  printf ("person\_copy = %s, %d \n", person\_copy.name, person\_copy.age );

  return 0;

}

Khác biệt giữa memcpy và strcpy:

strcpy() sao chép vùng nhớ từ vị trí mà source trỏ tới cho đến vị trí của kí tự ‘\0’ tới destination, do đó destination có giá trị chuỗi giống với source, còn với memcpy() thì copy số lượng bằng num byte theo chỉ định của người dùng.

## memcmp()

Cú pháp của hàm này như sau:

int memcmp(const void \* ptr1, const void \* ptr2, size\_t num);

Tác dụng: so sánh giá trị các vùng nhớ mà ptr1 và ptr2 trỏ tới theo từng byte, sẽ dừng lại khi so sánh đủ num byte.

Giá trị trả về:

* -1: Khi giá trị của byte đầu tiên không trùng của ptr1 nhỏ hơn ptr2.
* 0: Khi giá trị của 2 vùng nhớ bằng nhau
* 1: Khi giá trị của byte đầu tiên không trùng của ptr1 lớn hơn ptr2.

## Define

### Macro bằng define:

VD: định nghĩa số pi, tạo hàm tính diện tích

#define PI 3.14

#define area(r) (PI\*r\*r)

VD: tạo hàm hoán đổi vị trí swap

#define SWAP(type,x,y) do{type tmp = x; x = y; y = tmp;}while(0)

## Strstr(): tìm kiếm chuỗi con trong chuỗi

Cú pháp:

strstr(str, keyword);

Trong đó:

* str là chuỗi ban đầu
* keyword là chuỗi ký tự cần tìm trong chuỗi str.

Hàm strstr() sẽ trả về địa chỉ của vị trí đầu tiên tìm thấy của chuỗi ký tự keyword trong chuỗi str tính từ đầu chuỗi, và trả về vị trí này dưới dạng con trỏ chuỗi trong c.

VD

#include <stdio.h>

#include <string.h>

int main(void){

    char str[30] = "Good School";

    char \* p;

    p = strstr(str, "oo");

    printf("%s\n", p);

}

ood School

## Strcpy(): copy toàn bộ chuỗi

Cú pháp

strcpy(target, source);

Trong đó:

* targe là chuỗi đích
* source là chuỗi nguồn.

Hàm strcpy() sẽ copy toàn bộ nội dung của chuỗi source và phục chế nó vào chuỗi target. Lưu ý là hai chuỗi này cần phải tồn tại trước đó, thông qua việc [khởi tạo hay khai báo chuỗi trong C](http://laptrinhcanban.com/c/lap-trinh-c-co-ban/chuoi-string-trong-c/khai-bao-chuoi-trong-c).

VD

#include <stdio.h>

#include <string.h>

int main(void){

    char str1[30];

    char str2[30] = "I love C/C++";

    //Copy toàn bộ chuỗi str2 vào chuỗi str1

    strcpy(str1, str2);

    printf("%s\n", str1);

}

//I love C/C++

## Fget(): đọc từng dòng trong file

<http://laptrinhcanban.com/c/lap-trinh-c-co-ban/ham-trong-c/fgets-trong-c/>

## Pow(): tính luỹ thừa

Hàm pow trong C là một hàm có sẵn trong header file **math.h**, giúp chúng ta **tính luỹ thừa trong C**. Hàm pow sẽ tính luỹ thừa y của một số x và trả về kết quả.

Chúng ta sử dụng hàm pow để tính luỹ thừa trong C với cú pháp sau đây:

doule pow(doule x, doule y);

Trong đó x là cơ số, y là luỹ thừa, và doule là kiểu dữ liệu mặc định của cơ số và luỹ thừa sử dụng trong hàm pow.

Kết quả trả về của hàm pow sẽ là giá trị luỹ thừa mũ y của x với giá trị là xy.

VD

#include<stdio.h>

#include<math.h>

int main(){

    printf("Luy thua 2 cua 3 la: %lf\n", pow(2, 3));

    printf("Luy thua 3.1 cua 3 la: %f\n", pow(3.0, 3.1));

    printf("Luy thua 3.1 cua 3 la: %Lf\n", pow(2.012, 3.132));

    return 0;

}

Luy thua 2 cua 3 la: 8.000000

Luy thua 3.1 cua 3 la: 30.135326

Luy thua 3.1 cua 3 la: 0x1.1dd552283074ap+3

## Sqrt(): tính căn

Hàm sqrt trong C là một hàm có sẵn trong header file **math.h**, giúp chúng ta tính **căn bậc hai trong c**. Hàm sqrt sẽ nhận đối số là một số thực (kiểu double) và trả về căn bậc hai cũng thuộc kiểu double của số đó.

Chúng ta sử dụng hàm sqrt để tính căn bậc hai trong C với cú pháp sau đây:

double sqrt(double x);

Trong đó x là số thực thuộc kiểu double cần tìm căn bậc hai, và doule là kiểu dữ liệu mặc định của đối số cũng như giá trị trả về của hàm sqrt.

VD

#include <stdio.h>

#include <math.h>

int main(void){

    printf( "%lf\n", sqrt(4.0) );

    printf( "%lf\n", sqrt(100.0) );

    printf( "%lf\n", sqrt(0.0) );

    return 0;

}

2.000000

10.000000

0.000000

## Putchar():

Hàm putchar trong C là một hàm có sẵn trong header file **stdio.h**, giúp chúng ta **xuất 1 ký tự trong c**. Hàm putchar sẽ nhận đối số là một ký tự, và xuất ký tự đó ra màn hình.

Chúng ta sử dụng hàm putchar để **xuất 1 ký tự trong c** với cú pháp sau đây:

puts(char);

Trong đó char là ký tự (hoặc là biến chứa ký tự) mà chúng ta cần xuất ra màn hình.

Lưu ý là chúng ta chỉ có thể **xuất ra màn hình 1 ký tự** với hàm putchar(), do đó đối số sử dụng trong hàm cũng chỉ có thể là một ký tự hoặc là một biến có giá trị là một ký tự mà thôi.

VD

#include <stdio.h>

int main(void)

{

    putchar( 'a' );

    return 0;

}

a

## Abs(): tính giá trị tuyệt đối

Hàm abs trong C là một hàm có sẵn trong header file **stdlib.h**, giúp chúng ta **tính giá trị tuyệt đối trong c**. Hàm abs sẽ nhận đối số là một số nguyên (kiểu int) và trả về giá trị tuyệt đối của số đó.

Chúng ta sử dụng hàm abs để tính giá trị tuyệt đối trong C với cú pháp sau đây:

int abs(int x);

Trong đó x là số nguyên thuộc kiểu int cần tìm giá trị tuyệt đối.

Nếu x là số nguyên âm thì hàm abs sẽ trả về giá trị là số ngược dấu của x. Và nếu x là số 0 hoặc là số nguyên dương, hàm abs sẽ trả về giá trị là chính x.

#include <stdio.h>

#include <stdlib.h>

int main(void){

    printf("%d\n", abs(-12));

    printf("%d\n", abs(-3456));

    printf("%d\n", abs(5678));

    printf("%d\n", abs(0));

}

12

3456

5678

0

## Rand():

**Hàm rand trong c** là một hàm có sẵn trong header file **stdlib.h**, giúp chúng ta **tạo ra một số ngẫu nhiên** trong phạm vi từ 0 đến RAND\_MAX. Giá trị RAND\_MAX sẽ phụ thuộc vào môi trường lập trình C, ví dụ như trong VisualStudio thì giá trị này sẽ là 0x7fff, tương ứng với số thập phân là 2147483647.

VD

#include <stdio.h>

#include <stdlib.h>

int main(void){

    for (int i = 0;i < 10;i++) {

        printf("%d\n",rand());

    }

    return 0;

}

1804289383

846930886

1681692777

1714636915

1957747793

424238335

719885386

1649760492

596516649

1189641421

### Khắc phục nhược điểm các lần chạy lại chương trình sau, hàm rand cho các số ngẫu nhiên giống với hàm rand trước.

Đại khái là:

\\* -----

Sử dụng câu lệnh trước khi dùng rand()

srand((unsigned int)time(NULL));

-----------\*\

Chi tiết như sau:

Để có thể khắc phục nhược điểm của hàm rand và tạo số ngẫu nhiên khác nhau mỗi lần chạy hàm, chúng ta cần phải **thay đổi số ban đầu** được sử dụng để sinh ra các số ngẫu nhiên.

Để **thay đổi số ban đầu** sử dụng trong hàm rand, chúng ta sẽ cần dùng **hàm srand trong C** với cú pháp tổng quát sau đây:

srand(num);

Trong đó num chính là số ban đầu được sử dụng để tạo ra các số ngẫu nhiên trong các môi trường lập trình C.

Tuy nhiên mọi chuyện chưa dừng lại ở đây. Trong mỗi thời gian nhất định, số num này lại có giá trị giống nhau. Điều đó có nghĩa, nếu trong cùng 1 thời gian chạy hàm, thì các số ngẫu nhiên được tạo ra vẫn là giống nhau.

Để giải quyết vấn đề này, chúng ta cần phải tạo ra số num hoàn toàn khác nhau trong mỗi lần chạy **hàm srand trong c**. Và ý tưởng ở đây đó chính là đưa khái niệm thời gian vào hàm srand, nếu chúng ta **chỉ định thời khắc hiện tại** tính theo giây (s) làm đối số của hàm srand(), thì trong mỗi lần chạy hàm srand, thời khắc tính theo giây đều khác nhau, dẫn đến giá trị của num cũng khác nhau.

Để sinh ra thời khắc hiện tại, chúng ta sẽ dùng tời hàm time, và chúng ta sẽ sử dụng hàm rand kết hợp hàm time với cú pháp sau đây:

srand((unsigned int)time(NULL));

Lưu ý chúng ta cần include header file **time.h** để có thể sử dụng được hàm time trong chương trình.

Như vậy, chúng ta đã có thể dùng hàm srand để **thay đổi số ban đầu** để tạo ra các số ngẫu nhiên trong hàm rand.

Và chúng ta viết chương trình tạo số ngẫu nhiên khác nhau mỗi lần chạy hàm rand trong C như sau:

VD

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

//Tạo hàm GetRandom để chỉ định phạm vi sinh số ngẫu nhiên

int GetRandom(int min,int max){

    return min + (int)(rand()\*(max-min+1.0)/(1.0+RAND\_MAX));

}

int main(void){

    //Sử dụng hàm srand để thay đổi số nguồn sử dụng trong hàm rand

    srand((unsigned int)time(NULL));

    for (int i = 0;i < 10;i++) {

        printf("%d\n",GetRandom(1,6));

    }

    return 0;

}

Kết quả chạy lần 1

3

5

9

9

7

9

10

8

5

3

Kết quả chạy lần 2

10

1

2

5

6

5

8

5

9

7

# 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

# Fix bug:

## NULL undeclared

Thêm thư viện #include <[stddef.h](http://en.wikipedia.org/wiki/Stddef.h)>

# Tham khảo

1. Lập trình căn bản:

<https://laptrinhcanban.com/>

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