Lập trình C

# C là gì ?

C là ngôn ngữ lập trình bậc cao được phát triển ban đầu bởi Dennis M.Ritchie để phát triển hệ thống lập trình UNIX ở Bell Labs. C được phát triển ban đầu trên máy tính DEC PDP-11 năm 1972.

Năm 1978, Brian Kernighan và Dennis Ritchie công khai bản mô tả đầu tiên của ngôn ngữ C, được biết đến dưới tên tiêu chuẩn K&R.

Hệ điều hành UNIX, bộ biên dịch C, là tất cả những gì cần thiết cho việc viết các chương trình với ngôn ngữ C. Ngôn ngữ C hiện tại được sử dụng rộng rãi trong môi trường chuyên nghiệp vì những lý do sau đây:

* Dễ dàng trong việc học
* Ngôn ngữ có cấu trúc
* Nó cung cấp các chương trình hiệu quả.
* Nó có thể xử lý các hoạt động ở tầng thấp.
* Nó được biên dịch bởi nhiều nền tảng khác nhau.

Sự thật về ngôn ngữ C

* C được phát triển ban đầu để viết Hệ điều hành có tên UNIX.
* C là ngôn ngữ kế thừa của ngôn ngữ B được giới thiệu những năm 1970.
* Ngôn ngữ được chuẩn hóa năm 1988 bởi Viện tiêu chuẩn quốc gia Hoa Kỳ (ANSI).
* Hệ điều hành UNIX viết bởi ngôn ngữ C năm 1973.
* Ngày nay C được sử dụng rộng rãi nhất trong các ngôn ngữ lập trình hệ thống.
* Hầu hết các ứng dụng lớn đều có sự kế thừa, triển khai từ ngôn ngữ C.
* Hệ điều hành Linux và hệ quản trị cơ sở dữ liệu MySQL được viết bởi ngôn ngữ C.

Tại sao lại sử dụng C?

C được phát triển ban đầu cho việc phát triển hệ thống, đặc biệt là các hệ điều hành. C được thừa nhận như là một trong các ngôn ngữ phát triển hệ thống bởi nó cung cấp code và chạy một đoạn code một các nhanh chóng như các ngôn ngữ kiểu Assemly. Vài ví dụ về sử dụng C như sau:

* Hệ điều hành
* Bộ biên dịch các ngôn ngữ
* Các chương trình dịch mã số
* Các trình Text Editor (notepad ...)
* Các trình in ấn
* Network Drivers
* Các chương trình hiện đại
* Cơ sở dữ liệu
* Ngôn ngữ thông dịch
* Tiện ích

Chương trình C

Một chương trình C có thể thay đổi từ 3 dòng đến hàng triệu dòng code, và nên được viết trong một hoặc nhiều file với định dạng "**.c**", ví dụ **hello.c**. Bạn có thể sử dụng "**vi**", "**vim**" hoặc bất kỳ trình editor nào để viết chương trình C thành một file.

Bài hướng dẫn giả sử bạn đã biết cách sử dụng các trình soạn thảo và cách viết source code - mã nguồn bên trong một file chương trình.

# Hướng dẫn cài đặt C

## Cài đặt IDE để biên dịch và thực thi C

Có một số IDE có sẵn và miễn phí để biên dịch và thực thi các chương trình C. Bạn có thể chọn **Dev-C++, Code:: Blocks, hoặc Turbo C**. Tuy nhiên, lựa chọn phổ biến nhất và hay được sử dụng nhất là Dev-C++ và các chương trình C trong loạt bài này cũng được biên dịch và thực thi trong Dev-C++.

Bạn truy cập theo link sau để tải Dev-C++: [**Tải Dev-C++**](http://sourceforge.net/projects/orwelldevcpp/?source=typ_redirect). Trên trang này cũng bao gồm cả Code:: Blocks. Sau khi bạn tải xong, để cài đặt IDE này, bạn chỉ cần vào Google và gõ "cài đặt dev-c++" là có rất nhiều video hướng dẫn chi tiết, cho nên mình không cần trình bày thêm nữa.

Sau khi đã cài đặt xong, để biên dịch và thực thi một chương trình C, bạn: (a) vào **File -> New -> Project -> Console Application -> C project**, sau đó nhập tên vào hoặc (b) **File -> New -> Source File**. Cuối cùng, sao chép và dán chương trình C vào file bạn vừa tạo. Để biên dịch và thực thi, chọn **Execute -> Compile & Run**.

## Cài đặt để chạy trên Command Prompt

Nếu bạn muốn cài đặt để biên dịch và chạy trên Command Prompt, thì bạn nên đọc phần sau đây.

Nếu bạn đang muốn cài đặt chương trình C, bạn cần phải sử dụng 2 phần mềm trên máy tính của bạn: (a) Chương trình soạn văn bản - Text Editor và (b) Bộ biên dịch C.

## Text Editor

Được sử dụng để soạn thảo các chương trình. Ví dụ về một vài trình editor như Window Notepad, Notepad ++, vim hay vi…

Tên và các phiên bản của các trình editor có thể thay đổi theo các hệ điều hành. Ví dụ, Notepad được sử dụng trên Windows, hoặc vim hay vi được sử dụng trên Linux hoặc UNIX.

Các file bạn tạo trong trình editor được gọi là source file (file nguồn) và chứa các chương trình code. Các file trong chương trình C thường được đặt tên với phần mở rộng "**.c**".

Trước khi bắt đầu chương trình của bạn, hãy chắc chắn bạn có một trình editor trên máy tính và bạn có đủ kinh nghiệm để viết các chương trình máy tính, lưu trữ trong file và thực thi nó.

## Bộ biên dịch C

Mã nguồn được viết trong file nguồn dưới dạng có thể đọc được. Nó sẽ được biên dịch thành mã máy, để cho CPU có thể thực hiện các chương trình này dựa trên các lệnh được viết.

Bộ biên dịch được sử dụng để biên dịch mã nguồn (source code) của bạn đến chương trình có thể thực thi. Tôi giả sử bạn có kiến thức cơ bản về một bộ biên dịch ngôn ngữ lập trình.

Bộ biên dịch thông dụng nhất là bộ biên dịch GNU C/C++, mặt khác bạn có thể có các bộ biên dịch khác như HP hoặc Solaris với Hệ điều hành tương ứng.

Dưới đây là phần hướng dẫn giúp bạn cách cài đặt bộ biên dich GNU C/C++ trên các hệ điều hành khác nhau. Tôi đang đề cập đến C/C++ bởi vì bộ biên dịch GNU gcc hoạt động cho cả ngôn ngữ C và C++.

## Cài đặt trên môi trường UNIX/Linux

Nếu bạn đang sử dụng **Linux hoặc UNIX**, bạn có thể kiểm tra bộ GCC đã được cài đặt trên môi trường của bạn chưa bằng lệnh sau đây:

$ gcc -v

Nếu bạn có bộ cài đặt GNU trên máy tính của bạn, sau đó nó sẽ phản hồi một thông báo sau:

Using built-in specs.

Target: i386-redhat-linux

Configured with: ../configure --prefix=/usr .......

Thread model: posix

gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)

Nếu bộ GCC chưa được cài đặt, bạn có thể cài đặt nó với hướng dẫn trên đường link dưới đây: [**http://gcc.gnu.org/install/**](http://gcc.gnu.org/install/)

Bài hướng dẫn này được viết dựa trên Linux và tất cả các ví dụ dược biên dịch trên Cent OS của hệ thống Linux.

## Cài đặt trên môi trường Mac OS

Nếu bạn sử dụng hệ điều hành Mac OS X, cách đơn giản nhất để có GCC là download môi trường phát triển Xcode, bạn có thể sử dụng bộ biên dịch GNU cho C/C++.

Xcode được sẵn dưới link sau: [**developer.apple.com/technologies/tools/**](http://developer.apple.com/technologies/tools/).

## Cài đặt trên Windows

Để cài đặt GCC trên Windows bạn cần phải cài đặt MinGW. Để cài đặt MinGW, bạn truy cập vào [**www.mingw.org**](http://www.mingw.org/), và theo hướng dẫn trên trang download này. Download phiên bản mới nhất cho chương trình MinGW, dưới tên MinGW-<version>.exe.

Khi cài đặt MinWG, ít nhất bạn phải cài đặt gcc-core, gcc-g++, binutils và MinGW runtime, nhưng bạn có thể cài đặt nhiều hơn.

Thêm thư mục con bin trong nơi cài đặt MinGW vào biến môi trường **PATH** của bạn, bạn có thể sử dụng trực tiếp các công cụ dưới dạng command line một các dễ dàng.

Khi quá trình cài đặt hoàn tất, bạn có thể chạy gcc, g++, ar, ranlib, dlltool và các công cụ GNU khác trên Windows command line.

Cấu trúc chương trình C

ước khi chúng ta nghiên cứu về các khối tạo nên một chương trình C, đầu tiên bạn hãy xem một chương trình C mẫu.

Chương trình C: Hello World

Một chương trình C bao gồm những phần sau đây:

* Các lệnh tiền xử lý
* Các hàm
* Các biến
* Các lệnh và biểu thức
* Các comment

Đầu tiên hãy xem đoạn code đơn giản mà sẽ in ra màn hình hai từ "Hello World":

#include <stdio.h>

int main()

{

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

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

return 0;

}

Hãy xem các phần của chương trình bên trên:

1. Dòng đầu tiên của chương trình *#include <stdio.h>* là lệnh tiền xử lý, nhắc nhở bộ biên dịch C thêm tệp stdio.h trước khi biên dịch.
2. Dòng tiếp theo *int main()* là hàm main, nơi chương trình bắt đầu.
3. Dòng tiếp theo /\*...\*/ là dòng comment được bỏ qua bởi bộ biên dịch compiler và được dùng để thêm các chú thích cho chương trình. Đây được gọi là phần comment của chương trình.
4. Dòng tiếp theo *printf(...)* là một hàm chức năng khác của ngôn ngữ C , in ra thông điệp "Hello, World!" hiển thị trên màn hình.
5. Dòng tiếp theo **return 0;** kết thúc hàm chính và trả về giá trị 0.

Biên dịch & Thực thi Chương trình C

**1.** Nếu bạn sử dụng Dev-C++, Microsoft Visual Studio, ... thì bạn mở ứng dụng, vào File, chọn New để tạo một source file mới. Sau đó sao chép code trên vào source file mới này. Với Dev-C++, bạn sử dụng **Complile & Run** để biên dịch và chạy chương trình. Với Microsoft Visual Studio, bạn thực hiện tương tự và sau đó sử dụng **Build** để biên dịch và thực thi.

**2.** Nếu bạn dùng command prompt để biên dịch, bạn theo các bước sau:

1. Mở một trình editor và thêm dòng code ở trên.
2. Lưu tệp dưới dạng *hello.c*
3. Mở dòng nhắc lệnh và đi tới thư mục lưu trữ file.
4. Soạn thảo *gcc hello.c* và nhấn Enter để biên dịch dòng code trên.
5. Nếu không có lỗi trên đoạn code bên trên, dòng nhắc lệnh sẽ đưa bạn đến dòng tiếp theo và tạo ra file **a.out** có thể thực thi.
6. Bây giờ, soạn thảo *a.out* để thực hiện chương trình này.
7. Bây giờ bạn sẽ thấy dòng chữ "**Hello, World**" được in trên màn hình.

$ gcc hello.c

$ ./a.out

Hello, World!

Bạn phải chắc chắn bộ biên dịch gcc được cài đặt trên máy tính của bạn và bạn đang chạy nó trong thư mục chứa file nguồn hello.c.

# Cú pháp C cơ bản

Bạn đã biết về cấu trúc cơ bản của chương trình C, bây giờ bạn sẽ dễ dạng hiểu được những khối cơ bản trong ngôn ngữ C.

## Các Token trong C

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. Ví dụ, dòng lệnh C dưới đây bao gồm 5 token sau:

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

Các token riêng rẽ như sau:

printf

(

"Hello, World! \n"

)

;

## Dấu chấm phảy ; trong C

Chương trình C, dấu chấm phảy 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 dưới đây là 2 đoạn lệnh:

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

return 0;

## Comment trong C

Chú thích giống như việc trợ giúp trong chương trình C và được bỏ qua bởi bộ biên dịch. Nó bắt đầu với /\* và kết thúc với ký tự \*/ như dưới đây:

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

Bạn không thể có thêm một phần comment bên trong phần comment này.

## Định danh (Identifier) trong C

Một định danh 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. Dưới đây là một vài ví dụ định danh hợp lệ:

nam hoangminh abc ha\_noi a\_123

sinhvien \_hocphi j d23b5 nhanVien

## Các từ khóa trong C

Dưới đây là danh sách các từ khóa được dành riêng trong ngôn ngữ C. Các định danh hay biến, hằng số không thể đặt tên giống các từ khóa dưới đây, nếu không chương trình sẽ báo lỗi.

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |  |  |  |

## Khoảng trắng trong C

Một dòng có thể chứa khoảng trắng, có thể là những dòng comment, được biết đến như dòng trắng khi cùng được bộ biên dịch bỏ qua khi biên dịch.

Một khoảng trắng trong C có thể là một đoạn trống, tab, newline (dòng mới) hoặc comment. Một khoảng trắng chia một phần của lệnh thành nhiều phần và giúp bộ biên dịch phân biệt một thành phần trong một lệnh, như int , kết thúc thành phần và bắt đầu thành phần tiếp theo như lệnh sau:

int diemthi;

Phải có ít nhất một khoảng trắng ký tự giữa int và diemthi để bộ biên dịch hiểu và phân biệt được chúng. Mặt khác, xem lệnh dưới đây:

luong = luongcoban + phucap; // tinh tong luong

Không cần thiết khoảng trắng giữa luong và dấu =, hoặc giữa dấu = và luongcoban.

# Kiểu dữ liệu trong C

Các kiểu dữ liệu ám chỉ phần mở rộng của hệ thống được sử dụng cho khai báo biến với cái kiểu khác nhau. Kiểu của biến xác định lượng bộ nhớ được dùng để lưu biến đó và cách các bit được lưu trữ khi được thông dịch.

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). |

Các kiểu dữ liệu mảng và cấu trúc được sử dụng trong tập hợp như các kiểu dữ liệu gộp. Các kiểu là hàm chỉ định loại kiểu mà hàm trả về. Chúng ta sẽ xem các kiểu dữ liệu cơ bản ở phần dưới đây, trong đó những kiểu còn lại sẽ được nhắc đến ở các chương sau.

## Kiểu số nguyên (kiểu int) trong C

Bảng dưới đây đưa cho bạn những hiểu biết chi tiết về kiểu số nguyên với cỡ lưu trữ cũng như giới hạn của nó:

|  |  |  |
| --- | --- | --- |
| **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 |

Bạn có thể lấy cỡ chính xác của các kiểu của các biến trên những nền tảng cụ thể, bạn có thể sử dụng toán tử **sizeof**. Biểu thức **sizeof(kieu)** trả về cỡ của đối tượng hoặc kiểu dưới dạng byte. Dưới đây là ví dụ để lấy về size của đối tượng int trên bất kỳ máy tính nào.

#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

## Kiểu số thực dấu chấm động (Floating-Point) trong C

Bảng dưới đây đưa cho bạn những hiểu biết cụ thể về các kiểu số thực dấu chấm động tiêu chuẩn với cỡ lưu trữ và dải giá trị cũng như độ chính xác:

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

*float.h* trong Header file định nghĩa các macro cho phép bạn sử dụng các giá trị này và các kiểu cụ thể khác về giá trị biểu diễn nhị phân của số thực trong chương trình của bạn. Dưới đây là ví dụ sẽ in ra cỡ của kiểu float cũng như dải giá trị của nó:

#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:



## Kiểu 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. |

Bạn có thể không hiểu các điểm này về kiểu void, chúng ta nên tiếp tục và trong các chương tiếp theo, chúng ta sẽ nhắc lại về các điểm này.

Biến trong C

Một biến trong C không là gì nhưng là một tên được đưa ra đến bộ nhớ lưu trữ để chương trình có thể thao tác. Mỗi biến trong C có một kiểu xác định, để xác định cỡ và layout cho bộ nhớ biến đó. Phạm vi của giá trị có thể được dự trữ trong bộ nhớ, việc thiết lập các biểu thức có thể được áp dụng với biến.

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. Chữ hoa và chữ thường là hai đối tượng phân biệt bởi vì C là ngôn ngữ phân biệt **chữ hoa - chữ thường**. Dựa vào những loại cơ bản giải thích ở chương trước, có những loại kiểu của biến cơ bản như sau:

|  |  |
| --- | --- |
| **Kiểu** | **Miêu tả** |
| char | Là biến số nguyên, có kích cỡ 1 byte. |
| int | Là kiểu cho số tự nhiên. |
| float | Giá trị dấu chấm động độ chính xác đơn. |
| double | Giá trị dấu chấm động độ chính xác kép. |
| void | Đại diện cho loại không có kiểu. |

Ngôn ngữ lập trình C cho phép định nghĩa các loại kiểu biến khác nhau, có thể xem ở các chương sau như *biến liệt kê, biến con trỏ, biến mảng, biến cấu trúc, biến Union, …*.

Định nghĩa biến trong ngôn ngữ C

Định nghĩa biến nghĩa là thông báo với trình biên dịch nơi và cách tạo lưu trữ cho biến đó. Một định nghĩa biến xác định một kiểu dữ liệu và chứa danh sách của một hay nhiều biến của kiểu đó như sau:

kieu\_du\_lieu danh\_sach\_bien;

Ở đây, **kieu\_du\_lieu** là của kiểu dữ liệu của ngôn ngữ C như char, w\_char, int, float, double, bool hay bất kỳ kiểu đối tượng được người dùng định nghĩa… **danh\_sach\_bien** có thể bao gồm một hoặc nhiều tên định danh ngăn cách nhau bởi dấu phảy. Vài ví dụ khai báo hợp lệ của biến như sau:

int i, j;

char ho, ten, c, ch;

float f, luong, diemthi;

double d;

Dòng **int i, j;** vừa khai báo và định nghĩa cho biến i, j, k và hướng dẫn trình biên dịch để tạo các biến dưới tên i, j, k với kiểu int.

Biến có thể được khởi tạo (được gán các giá trị ban đầu) trong khai báo của nó. Một phần khởi tạo bao gồm một dấu "=" theo sau bởi một biểu thức hằng số số như sau:

kieu\_du\_lieu ten\_bien = gia\_tri;

Vài ví dụ dưới đây:

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'.

Với định nghĩa không có giá trị khởi tạo, biến static có thể lưu trữ với giá trị NULL, (tất cả các byte có giá trị 0), giá trị ban đầu của tất cả các biến của tất cả các kiểu khác có giá trị không xác định.

Khai báo biến trong ngôn ngữ C:

Khai báo biến cung cấp một sự bảo đảm cho trình biên dịch nhận biết rằng không có biến nào với kiểu và tên giống nó được khai báo trước đó, nếu không sẽ xảy ra lỗi ở quá trình biên dịch. Một khai báo biến chỉ có ý nghĩa ở thời gian biên dịch, trình biên dịch cần khai báo biến cụ thể tại thời gian nối với chương trình.

Một khai báo biến rất hữu dụng khi bạn sử dụng đồng thời nhiều file và bạn định nghĩa biến của bạn ở một trong những file đó. Bạn có thể sử dụng từ khóa **extern** để khai báo biến ở bất kì nơi đâu. Do đó bạn có thể khai báo một biến nhiều lần trong chương trình C nhưng chỉ phải định nghĩa trong một file, một hàm hay một khối code.

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

Thử ví dụ dưới đây, nơi biến được khai báo ở trên đầu, nhưng chúng được định nghĩa và khởi tạo trong hàm main:

#include <stdio.h>

// phan khai bao bien:

extern int a, b;

extern int c;

extern float f;

int main ()

{

/\* phan dinh nghia bien: \*/

int a, b;

int c;

float f;

/\* phan khoi tao gia tri thuc su \*/

a = 15;

b = 35;

c = a + b;

printf("Gia tri cua c la : %d \n", c);

f = 50.0/3.0;

printf("Gia tri cua f la : %f \n", f);

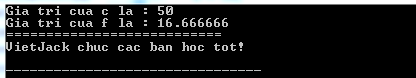
printf("===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Vài khái niệm có thể áp dụng trong khai báo hàm nơi bạn cung cấp tên hàm ở thời điểm nó khai báo và định nghĩa nó ở bất kì nơi đâu. Ví dụ:

// phan khai bao ham

int tenham();

int main()

{

// loi goi ham

int i = tenham();

}

// phan dinh nghia ham

int tenham()

{

return 0;

}

Lvalue và Rvalue trong C:

Có hai kiểu Expression:

1. **lvalue :** Expression mà chỉ tới vị trí bộ nhớ là "**lvalue**". Một lvalue có thể xuất hiện hoặc bên trái hoặc bên phải của một phép gán.
2. **rvalue :** Liên quan tới giá trị dữ liệu được lưu trữ tại một số địa chỉ trong bộ nhớ. Một rvalue là một expression mà không thể có một giá trị được gán tới nó, nghĩa là một rvalue có thể xuất hiện ở bên phải nhưng không phải bên trái của một phép gán.

*Các biến* là các lvalue và thường xuất hiện ở cạnh trái của phép gán. *Các hằng số là số* là rvalue và không thể được gán và không thể xuất hiện bên cạnh trái của phép gán. Dưới đây là một khai báo hợp lệ:

int g = 20;

Nhưng sau đây là một khai báo không hợp lệ và sẽ có thông báo lỗi:

10 = 20;

Hằng số trong C

Hằng số (constant) hướng đến những giá trị cố định mà chương trình không thể thay đổi trong quá trình thực thi. Những giá trị cố định đó cũng được gọi là **literals**.

Hằng số có thể là một kiểu dữ liệu bất kỳ nào như kiểu dữ liệu: *số nguyên, số thực, ký tự hay chuỗi*. Có những hằng số kiểu liệt kê (enumeration).

Một **hằng số** có thể được coi như một biến thường ngoài việc giá trị của nó không thể thay đổi sau khi được định nghĩa.

Hằng số nguyên trong C

Giá trị hằng số nguyên có thể là hệ thập phân (decimal), hệ bát phân (octal) hoặc hệ thập lục phân (hexadecimal). Tiền tố (prefix) xác định cơ bản hay cơ số: 0x hay 0X cho kiểu hexadecimal (hệ 16), 0 cho octal (hệ 8), và không có gì là hệ thập phân.

Một giá trị hằng số có thể có phần hậu tố (suffix) là sự kết hợp của **U và L, cho kiểu Unsigned và kiểu Long**. Phần kết thúc có thể là chữ hoa hoặc chữ thường theo bất cứ thứ tự nào.

Dưới đây là ví dụ cho kiểu hằng số nguyên:

212 /\* la hop le \*/

215u /\* la hop le \*/

0xFeeL /\* la hop le \*/

078 /\* Khong hop le: 8 khong la ky so trong he bat phan (octal) \*/

032UU /\* Khong hop le: ban khong the lap lai hau to (suffix) \*/

Dưới đây là các ví dụ khác với một vài cách khai báo với kiểu số nguyên:

85 /\* he thap phan \*/

0213 /\* he bat phan (octal) \*/

0x4b /\* he thap luc phan (hexadecimal) \*/

30 /\* int \*/

30u /\* unsigned int \*/

30l /\* long \*/

30ul /\* unsigned long \*/

Hằng số thực trong C

Một hằng số thực dấu chấm động có một phần nguyên, một giá trị decimal, phần phân số và phần mũ. Bạn có thể biểu diễn giá trị dấu chấm động trong kiểu thập phân và kiểu phân số.

Khi biểu diễn giá trị với định dạng thập phân, bạn phải thêm phần integer, phần mũ hoặc cả hai. Phần mũ được viết bởi e hoặc E.

Dưới đây là vài ví dụ cho phần dấu chấm động:

3.14159 /\* Hop le \*/

314159E-5L /\* Hop le \*/

510E /\* Khong hop le: phan mu chua hoan thien \*/

210f /\* Khong hop le: khong co phan decimal va phan mu \*/

.e55 /\* Khong hop le: thieu phan phan so va phan nguyen \*/

Hằng ký tự trong C

Phần ký tự được đóng mở trong dấu nháy đơn ('), ví dụ 'x' và có thể được lưu trữ trong một biến đơn giản kiểu **char**.

Một ký tự có thể là một ký tự thường (ví dụ 'x') hoặc chuỗi thoát (vd: '\t'), hoặc một ký tự phổ thông (vd: '\u02C0').

Có những ký tự cụ thể trong C khi bắt đầu bằng dấu \ sẽ có ý nghĩa đặc biệt và được dùng để biểu diễn dòng mới (\n), tab mới (\t). Dưới đây là danh sách các ký tự đặc biệt:

|  |  |
| --- | --- |
| **Dãy thoát** | **Ý nghĩa** |
| \\ | Ký tự \ |
| \' | Ký tự ' |
| \" | Ký tự " |
| \? | Ký tự ? |
| \a | Tiếng chuông |
| \b | Backspace |
| \f | Form feed |
| \n | Dòng mới |
| \r | Carriage return |
| \t | tab ngang |
| \v | tab dọc |
| \ooo | Số trong cơ số 8 của 1 đến 3 chữ số |
| \xhh . . . | Số thập lục phân của một hoặc nhiều chữ số |

Sau đây là ví dụ để chỉ một số ký tự dãy thoát:

#include <stdio.h>

int main()

{

printf("Hoc\tLap\tTrinh\tC\tTai\tVietjack\n\n");

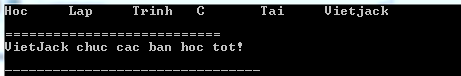
printf("===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Hằng chuỗi trong C

Hằng chuỗi được bao với dấu ngoặc kép "". Một chuỗi bao gồm các ký tự tương tự với hằng ký tự: *ký tự thuần, chuỗi thoát và ký tự phổ thông*.

Bạn có thể chia các dòng dài thành nhiều dòng sử dụng các giá trị chuỗi và ngăn cách chúng bởi các khoảng trắng.

Dưới đây là một vài ví dụ với hằng số chuỗi. Ba chuỗi dưới đây có giá trị giống nhau:

"hello, vietnam"

"hello, \

vietnam"

"hello, " "v" "ietnam"

Định nghĩa hằng số trong C

Có 2 cách đơn giản trong C để định nghĩa hằng số:

1. Sử dụng bộ tiền xử lý **#define**.
2. Sử dụng từ khóa **const**.

Sử dụng bộ tiền xử lý #define trong C

Dưới đây là mẫu để sử dụng bộ tiền xử lý #define để định nghĩa một hằng số:

#define dinh\_danh gia\_tri

Dưới đây là ví dụ chi tiết:

#include <stdio.h>

#define CHIEUDAI 15

#define CHIEURONG 12

#define NEWLINE '\n'

int main()

{

int dientich;

dientich = CHIEUDAI \* CHIEURONG;

printf("Dien tich hinh chu nhat la: %d", dientich);

printf("%c", NEWLINE);

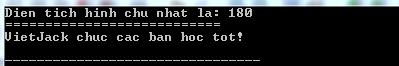
printf("===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Sử dụng từ khóa const trong C

Bạn có thể sử dụng tiền tố **const** để khai báo các hằng số với một kiểu cụ thể như sau:

const kieu\_du\_lieu ten\_bien = gia\_tri;

Dưới đây là ví dụ chi tiết:

#include <stdio.h>

int main()

{

const int CHIEUDAI = 15;

const int CHIEURONG = 12;

const char NEWLINE = '\n';

int dientich;

dientich = CHIEUDAI \* CHIEURONG;

printf("Dien tich hinh chu nhat la: %d", dientich);

printf("%c", NEWLINE);

printf("===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

Biên dịch và chạy chương trình C trên sẽ cho kết quả giống như trên.

Ghi chú trong thực tế lập trình chúng ta thường đặt tên hằng là**CHỮ HOA**.

Lớp lưu trữ trong C

Một lớp lưu trữ định nghĩa phạm vi (nhìn thấy được) và thời gian tồn tại của một biến hoặc/và các hàm trong chương trình C. Những đặc tả đi trước kiểu có thể được thay đổi. Dưới đây là những lớp lưu trữ, có thể sử dụng trong chương trình C.

* auto
* register
* static
* extern

Lớp lưu trữ auto trong C

Lớp lưu trữ **auto** là lớp lưu trữ mặc định cho tất cả các biến cục bộ:

{

int diemthi;

auto int diemthi;

}

Ví dụ trên để định nghĩa hai biến trong cùng một lớp lưu trữ, auto có thể được sử dụng bên trong hàm, ví dụ biến cục bộ.

Lớp lưu trữ register trong C

Lớp lưu trữ **register** có thể được sử dụng để định nghĩa biến cục bộ và có thể được lưu trữ trong một vùng đăng ký thay vì RAM. Điều này nghĩa là biến này có cỡ tối đa tương đương với cỡ đăng ký.

{

register int hocphi;

}

Lớp lưu trữ register chỉ được sử dụng cho biến mà yêu cầu truy cập nhanh như bộ đếm. Chú ý rằng định nghĩa 'register' không đồng nghĩa với biến đó có thể lưu trữ trong một thanh ghi. Nó nghĩa là có thể lưu trữ trong thanh ghi phụ thuộc vào phần cứng và với các hạn chế nhất định.

Lớp lưu trữ static trong C

Lớp lưu trữ **static** hướng dẫn trình biên dịch giữ các giá trị biến **cục bộ** tồn tại trong thời gian sống của chương trình thay vì việc tạo ra và hủy nó mỗi lần chạy qua phạm vi đó. Do đó, tạo một biến **cục bộ static** cho phép chúng lưu trữ các giá trị với các hàm gọi.

Lớp static này có thể được áp dụng cho biến toàn cục. Khi việc này diễn ra, nó gây ra phạm vi của biến được giới hạn trong file mà nó khai báo.

Trong lập trình C, khi static được sử dụng ứng với lớp, nó dẫn đến chỉ có một bản copy của lớp khai báo được chia sẻ bởi tất cả các đối tượng sử dụng lớp này.

#include <stdio.h>

/\* phan khai bao ham \*/

void ham(void);

static int biendem = 4; /\* day la bien toan cuc \*/

main()

{

while(biendem--)

{

ham();

}

printf("===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

/\* phan dinh nghia ham \*/

void ham( void )

{

static int i = 6; /\* bien cuc bo static \*/

i++;

printf("i co gia tri la %d va biendem co gia tri la %d\n", i, biendem);

}

Bạn có thể chưa hiểu ví dụ này này bởi vì bạn có sử dụng biến *toàn cục*, sẽ được giới thiệu ở bài tới. Biên dịch và chạy chương trình C trên sẽ cho kết quả:

Lớp lưu trữ extern trong C

Lóp lưu trữ **extern** được sử dụng để đưa tham chiếu đến các biến toàn cục mà được nhìn thấy bởi tất cả các file chương trình.

Khi bạn có nhiều file và bạn định nghĩa các biến cục bộ hoặc hàm, nó sẽ được sử dụng ở các file khác. Để hiểu vấn đề này, *extern* được sử dụng để khai báo biến *toàn cục* hoặc hàm ở file khác.

Từ khóa extern được sử dụng khi hai hoặc nhiều file chia sẻ chung một biến hoặc hàm như ví dụ dưới đây:

**File đầu tiên: file1.c**

#include <stdio.h>

int biendem ;

extern void ham\_extern();

main()

{

biendem = 5;

ham\_extern();

}

**File thứ 2: file2.c**

#include <stdio.h>

extern int biendem;

void ham\_extern(void)

{

printf("biendem co gia tri la %d\n", biendem);

}

Tại đây, *extern* là từ khóa được sử dụng để khai báo *biendem* ở dòng thứ hai nơi nó được định nghĩa ở file thứ nhất, main.c . Bây giờ, nếu bạn đang sử dụng command prompt, bạn biên dịch 2 file như sau:

$gcc main.c support.c

Nó sẽ cung cấp chương trình thực thi **a.out**, khi chương trình này được chạy sẽ in ra kết quả sau đây:

5

Toán tử trong C

Toán tử là một biểu tượng mà nói với trình biên dịch thực hiện một phép toán học nhất định hoặc thao tác hợp lý. Ngôn ngữ C có sẵn rất nhiều toán tử và cung cấp các kiểu toán tử sau đây:

* Toán tử số học
* Toán tử quan hệ
* Toán tử logic
* Toán tử so sánh bit
* Toán tử gán
* Toán tử hỗn hợp

Bài hướng dẫn sẽ giải thích toán tử số học, quan hệ, logic, so sánh bit, gán và các toán tử khác, từng loại một.

Toán tử số học trong C

Bảng dưới đây chỉ ra tất cả các toán tử số học được hỗ trợ bởi ngôn ngữ C. Giả sử biến **A** có giá trị 10 và biến **B** có giá trị 20:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| + | Thêm hai toán hạng | A + B sẽ cho kết quả là 30 |
| - | Trừ giá trị toán hạng hai từ toán hạng đầu | A - B sẽ cho kết quả là -10 |
| \* | Nhân hai toán hạng | A \* B sẽ cho kết quả là 200 |
| / | Chia lấy phần nguyên hai toán hạng | B / A sẽ cho kết quả là 2 |
| % | Chia lấy phần dư | B % A sẽ cho kết quả là 0 |
| ++ | Lượng gia giá trị toán hạng thêm 1 đơn vị | A++ sẽ cho kết quả là 11 |
| -- | Lượng giảm giá trị toán hạng một đơn vị | A-- sẽ cho kết quả là 9 |

Toán tử quan hệ

Bảng dưới đây chỉ ra tất cả các toán tử quan hệ được hỗ trợ bởi ngôn ngữ C. Giả sử rằng biến **A** có giá trị 10 và biến **B** có giá trị 20, ta có:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| == | Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true. | (A == B) là không đúng. |
| != | Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true. | (A != B) là true. |
| > | Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true. | (A > B) là không đúng. |
| < | Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true. | (A < B) là true. |
| >= | Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true. | (A >= B) là không đúng. |
| <= | Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true. | (A <= B) là true. |

Toán tử logic

Bảng dưới đây chỉ rõ tất cả các toán tử logic được hỗ trợ bởi ngôn ngữ C. Giả sử biến **A** có giá trị 1 và biến **B** có giá trị 0:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| && | Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true. | (A && B) là false. |
| || | Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true. | (A || B) là true. |
| ! | Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false. | !(A && B) là true. |

Toán tử so sánh bit

Toán tử so sánh bit làm việc trên đơn vị bit, tính toán biểu thức so sánh từng bit. Bảng dưới đây về &, |, và ^ như sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Giả sử nếu A = 60; và B = 13; thì bây giờ trong định dạng nhị phân chúng sẽ là như sau:

A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011

Các toán tử so sánh bit được hỗ trợ bởi ngôn ngữ C được liệt kê trong bảng dưới đây. Giá sử ta có biến A có giá tri 60 và biến B có giá trị 13, ta có:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| & | Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng. | (A & B) sẽ cho kết quả là 12, tức là 0000 1100 |
| | | Toán tử OR (hoặc) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng. | (A | B) sẽ cho kết quả là 61, tức là 0011 1101 |
| ^ | Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai. | (A ^ B) sẽ cho kết quả là 49, tức là 0011 0001 |
| ~ | Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại). | (~A ) sẽ cho kết quả là -61, tức là 1100 0011. |
| << | Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải. | A << 2 sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit) |
| >> | Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải. | A >> 2 sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit) |

Toán tử gán

Đây là những toán tử gán được hỗ trợ bởi ngôn ngữ C:

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| = | Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái. | C = A + B sẽ gán giá trị của A + B vào trong C |
| += | Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái. | C += A tương đương với C = C + A |
| -= | Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái. | C -= A tương đương với C = C - A |
| \*= | Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái. | C \*= A tương đương với C = C \* A |
| /= | Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái. | C /= A tương đương với C = C / A |
| %= | Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái. | C %= A tương đương với C = C % A |
| <<= | Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải. | C <<= 2 tương đương với C = C << 2 |
| >>= | Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải. | C >>= 2 tương đương với C = C >> 2 |
| &= | Phép AND bit | C &= 2 tương đương với C = C & 2 |
| ^= | Phép OR loại trừ bit | C ^= 2 tương đương với C = C ^ 2 |
| |= | Phép OR bit. | C |= 2 tương đương với C = C | 2 |

Các toán tử hỗn hợp ↦ sizeof & ternary

Có một số toán tử hỗn hợp quan trọng là **sizeof** và **? :** được hỗ trợ bởi ngôn ngữ C.

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| sizeof() | Trả lại kích cỡ của một biến | sizeof(a), với a là integer, thì sẽ trả lại kết quả là 4. |
| & | Trả lại địa chỉ của một biến. | &a; sẽ cho địa chỉ thực sự của biến a. |
| \* | Trỏ tới một biến. | \*a; sẽ trỏ tới biến a. |
| ? : | Biểu thức điều kiện | Nếu điều kiện là true ? thì giá trị X : Nếu không thì giá trị Y |

Thứ tự ưu tiên toán tử trong C

Thứ tự ưu tiên toán tử trong C xác định cách biểu thức được tính toán. Ví dụ, toán tử nhân có quyền ưu tiên hơn toán tử cộng, và nó được thực hiện trước.

Ví dụ, x = 7 + 3 \* 2; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử \* có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân 3 \* 2 và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất thì ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

|  |  |  |
| --- | --- | --- |
| **Loại** | **Toán tử** | **Thứ tự ưu tiên** |
| Postfix | () [] -> . ++ - - | Trái sang phải |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Phải sang trái |
| Tính nhân | \* / % | Trái sang phải |
| Tính cộng | + - | Trái sang phải |
| Dịch chuyển | << >> | Trái sang phải |
| Quan hệ | < <= > >= | Trái sang phải |
| Cân bằng | == != | Trái sang phải |
| Phép AND bit | & | Trái sang phải |
| Phép XOR bit | ^ | Trái sang phải |
| Phép OR bit | | | Trái sang phải |
| Phép AND logic | && | Trái sang phải |
| Phép OR logic | || | Trái sang phải |
| Điều kiện | ?: | Phải sang trái |
| Gán | = += -= \*= /= %=>>= <<= &= ^= |= | Phải sang trái |
| Dấu phảy | , | Trái sang phải |

# Điều khiển luồng trong C

[Trang trước](https://vietjack.com/lap_trinh_c/toan_tu_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/vong_lap_trong_c.jsp)

Các cấu trúc điều khiển luồng yêu cầu lập trình viên xác định một hoặc nhiều điều kiện để đánh giá và kiểm tra bởi chương trình, cùng với các lệnh được thực hiện nếu điều kiện được xác định là đúng, hoặc các lệnh khác được thực hiện nếu điều kiện xác định là sai.

Dưới đây là mẫu chung của một cấu trúc điều khiển luồng hay gặp trong ngôn ngữ lập trình.



Ngôn ngữ lập trình C giả sử rằng mọi giá trị **không phải zero** hoặc **không phải null** đều có giá trị **true**, nếu có giá trị **zero hoặc null**, nó có giá trị **false**.

Ngôn ngữ C cung cấp các kiểu cấu trúc điều khiển luồng sau. Bạn nhấn vào đường link để biết thêm chi tiết về chúng.

|  |  |
| --- | --- |
| **Lệnh** | **Miêu tả** |
| [**Lệnh if**](https://vietjack.com/lap_trinh_c/lenh_if_trong_c.jsp) | Một lệnh **if** bao gồm một biểu thức logic theo sau bởi một hoặc nhiều lệnh khác. |
| [**Lệnh if...else**](https://vietjack.com/lap_trinh_c/lenh_if_else_trong_c.jsp) | Một lệnh **if** có thể theo sau bởi một lệnh **else** (tùy ý: có hoặc không), mà có thể được thực hiện khi biểu thức logic có giá trị false. |
| [**Lồng lệnh if**](https://vietjack.com/lap_trinh_c/long_lenh_if_trong_c.jsp) | Bạn có thể sử dụng lệnh **if** hoặc **else if** bên trong lệnh **if** hoặc **else if** khác. |
| [**Lệnh switch**](https://vietjack.com/lap_trinh_c/lenh_switch_trong_c.jsp) | Một lệnh **switch** cho phép kiểm tra điều kiện của một biến trước khi thực thi các lệnh. |
| [**Lồng lệnh switch**](https://vietjack.com/lap_trinh_c/long_lenh_switch_trong_c.jsp) | Bạn có thể sử dụng một lệnh **switch** bên trong một lệnh **switch** khác. |

## Toán tử điều kiện ? : trong C

Chúng ta đã bàn về **toán tử điều kiện ? :** trong chương trước mà có thể được dùng để đổi vị trí cho lệnh **if...else**. Nó có mẫu chung như sau:

bieuthuc1 ? bieuthuc2 : bieuthuc3;

Trong đó Exp1, Exp2 và Exp3 là các biểu thức. Chú ý việc sử dụng và đặt của dấu hai chấm.

Giá trị của biểu thức Exp1 trước dấu ? có giá trị **true**, Exp2 được thực hiện, và giá trị của nó là giá trị của biểu thức. Nếu Exp1 là **false** thì Exp3 được thực hiện và giá trị của nó là giá trị của biểu thức.

# Vòng lặp trong C

[Trang trước](https://vietjack.com/lap_trinh_c/dieu_khien_luong_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/ham_trong_c.jsp)

Có một tình huống mà bạn cần phải thực hiện một đoạn code một vài lần. Nhìn chung, các câu lệnh được thực hiện một cách tuần tự. Câu lệnh đầu tiên của hàm được thực hiện trước, sau đó đến câu thứ 2 và tiếp tục.

Ngôn ngữ lập trình cung cấp cho chúng ta nhiều cấu trúc điều khiển và cho phép bạn thực hiện những phần phức tạp.

Vòng lặp cho phép thực hiện một lệnh và một nhóm lệnh nhiều lần , dưới đây là dạng tổng quát:



C hỗ trợ những lệnh điều khiển sau đây. Click chuột vào link để xem chi tiết.

|  |  |
| --- | --- |
| **Kiểu vòng lặp** | **Miêu tả** |
| [**Vòng lặp While**](https://vietjack.com/lap_trinh_c/vong_lap_while_trong_c.jsp) | Lặp lại một hoặc một nhóm các lệnh trong khi điều kiện đã cho là đúng. Nó kiểm tra điều kiện trước khi thực hiện thân vòng lặp. |
| [**Vòng lặp for**](https://vietjack.com/lap_trinh_c/vong_lap_for_trong_c.jsp) | Thực thi một dãy các lệnh nhiều lần và tóm tắt đoạn code mà quản lý biến vòng lặp. |
| [**Vòng lặp do...while**](https://vietjack.com/lap_trinh_c/vong_lap_do_while_trong_c.jsp) | Giống lệnh While, ngoại trừ ở điểm là nó kiểm tra điều kiện ở cuối thân vòng lặp. |
| [**Lồng các vòng lặp**](https://vietjack.com/lap_trinh_c/long_vong_lap_trong_c.jsp) | Bạn có thể sử dụng một hoặc nhiều vòng lặp trong các vòng lặp while, for hoặc do..while khác. |

## Các lệnh điều khiển vòng lặp

Các lệnh điều khiển vòng lặp thay đổi sự thực thi lệnh từ dãy thông thường của nó. Khi sự thực thi lệnh rời khỏi một phạm vi, tất cả các đối tượng tự động mà được tạo ra trong phạm vi đó bị hủy.

C hỗ trợ các lệnh điều khiển vòng lặp sau đây. Click vào các đường link sau để biết thêm chi tiết.

|  |  |
| --- | --- |
| **Lệnh điều khiển** | **Miêu tả** |
| [**Lệnh break**](https://vietjack.com/lap_trinh_c/lenh_break_trong_c.jsp) | Kết thúc **vòng lặp** hoặc lệnh **switch** và chuyển sang thực thi vòng lặp hoặc lệnh switch ngay sau nó. |
| [**Lệnh continue**](https://vietjack.com/lap_trinh_c/lenh_continue_trong_c.jsp) | Khi gặp lệnh này thì chương trình sẽ bỏ qua các câu lệnh ở dưới nó (trong cùng một câu lệnh lặp) để thực hiện vòng lặp mới. |
| [**Lệnh goto**](https://vietjack.com/lap_trinh_c/lenh_goto_trong_c.jsp) | Chuyển tới lệnh được gán. Mặc dù vậy, nó được khuyên rằng không nên sử dụng lệnh goto trong chương trình của bạn. |

## Vòng lặp vô hạn

Một vòng lặp là vòng lặp vô hạn khi một điều kiện không bao giờ false. Vòng lặp **for** thường được sử dụng cho mục đích này. Khi bạn để ba biểu thức điều kiện trong vòng lặp *for* trống thì bạn sẽ tạo ra một vòng lặp vô hạn.

#include <stdio.h>

int main ()

{

for( ; ; )

{

printf("Vong lap nay se chay mai mai.\n");

}

return 0;

}

Khi biểu thức điều kiện vắng mặt, nó được giả sử là luôn đúng.

**Ghi chú:** Bạn có thể dừng (kết thúc) một vòng lặp vô hạn bởi nhấn tổ hợp phím Ctrl + C.

Hàm trong C

[Trang trước](https://vietjack.com/lap_trinh_c/vong_lap_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/bien_toan_cuc_va_bien_cuc_bo_trong_c.jsp)

Một hàm là một nhóm các lệnh đi cùng nhau để thực hiện một nhiệm vụ. Mỗi chương trình C có ít nhất một hàm là hàm **main()**, và tất cả hầu hết các chương trình bình thường đều định nghĩa thêm các hàm.

Bạn có thể chia đoạn code của bạn thành những hàm riêng biệt. Cách bạn chia đoạn code của bạn thành các hàm khác nhau phụ thuộc vào bạn, nhưng theo tính logic, một hàm thường có một nhiệm vụ nhất định.

Một sự **khai báo** hàm thông báo với bộ biên dịch về tên của hàm, kiểu trả về và tham số. Một **định nghĩa** hàm cung cấp phần thân của một hàm.

Các thư viện tiêu chuẩn của ngôn ngữ C cung cấp rất nhiều hàm có sẵn để chương trình của bạn có thể gọi. Ví dụ, hàm **strcat()** có thể nối hai đoạn chuỗi, hàm **memcpy()** dùng để copy một vùng nhớ đến một vùng nhớ khác và rất nhiều hàm khác nữa.

Một hàm được biết đến với các tên khác nhau như một phương thức, một tuyến phụ hoặc một thủ tục.

Định nghĩa một hàm trong C

Mẫu chung của định nghĩa hàm trong Ngôn ngữ C như sau:

kieu\_tra\_ve ten\_ham( danh\_sach\_tham\_so )

{

than\_cua\_ham

}

Một định nghĩa hàm trong ngôn ngữ C bao gồm *đầu hàm* và một *thân hàm*. Dưới đây là các phần của một hàm:

* **Kiểu trả về**: Một hàm có thể trả về một giá trị. **Kieu\_tra\_ve** là dạng dữ liệu của giá trị mà hàm trả về. Vài hàm cung cấp các hoạt động và không trả về giá trị nào cả. Đó là hàm **void**.
* **Tên hàm:** Đây là tên thực sự của hàm. Tên hàm và danh sách tham số cấu tạo nên dấu hiệu hàm.
* **Tham số:** Khi hàm được gọi, bạn phải truyền vào danh sách các tham số. Một giá trị hướng đến một tham số thực tế. Danh sách tham số có các kiểu, thứ tự và số lượng các tham số của hàm. Các tham số trong hàm là tùy chọn, nghĩa là một hàm có thể không có tham số.
* **Thân hàm:** Phần thân của một hàm bao gồm tập hợp các lệnh xác định những gì mà hàm thực hiện.

Ví dụ:

Sau đây là mã nguồn cho một hàm có tên gọi là **timGTLN()**. Hàm này có 2 tham số: so1 và so2 và trả về giá trị lớn nhất giữa hàm số:

/\* ham tra ve gia tri lon nhat giua hai so \*/

int timGTLN(int so1, int so2)

{

/\* khai bao bien cuc bo \*/

int ketqua;

if (so1 > so2)

ketqua = so1;

else

ketqua = so2;

return ketqua;

}

Khai báo hàm trong C

Một **khai báo** hàm thông báo cho trình biên dịch về tên hàm và cách gọi của hàm. Phần thân hàm có thể định nghĩa một cách rời rạc.

Một khai báo hàm có các phần sau đây:

kieu\_tra\_ve ten\_ham( danh sach tham so );

Ví dụ khi định nghĩa hàm timGTLN(), dưới đây là câu khai báo hàm:

int timGTLN(int so1, int so2);

Tên các tham số không quan trọng trong việc khai báo hàm, những kiểu dưới đây là cách khai báo hợp lệ:

int timGTLN(int, int);

Một khai báo hàm được yêu cầu khi bạn định nghĩa một hàm và mã nguồn và khi gọi một hàm từ một file nguồn khác. Trong trường hợp này, bạn nên khai báo hàm trước khi gọi hàm đó.

Gọi hàm trong C

Trong khi tạo một hàm, bạn định nghĩa những gì hàm phải làm. Để sử dụng một hàm, bạn phải gọi hàm đó để thực hiện một nhiệm vụ cụ thể.

Khi một chương trình gọi một hàm, phần điều khiển được chuyển đến hàm được gọi. Một hàm được gọi thực hiện các nhiệm vụ được định nghĩa và trả về giá trị sau khi thực hiện chương trình.

Để gọi hàm, bạn đơn giản cần truyền các tham số được yêu cầu cùng với tên của hàm và nếu hàm trả về các giá trị, bạn có thể dự trữ các giá trị trả về này, ví dụ:

#include <stdio.h>

/\* khai bao ham \*/

int timGTLN(int so1, int so2);

int main ()

{

/\* phan dinh nghia bien cuc bo \*/

int a = 667;

int b = 7028;

int kq;

/\* goi ham de tim gia tri lon nhat \*/

kq = timGTLN(a, b);

printf( "Gia tri lon nhat la : %d\n", kq );

printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

/\* phan dinh nghia ham de tra ve gia tri lon nhat giua hai so \*/

int timGTLN(int so1, int so2)

{

/\* khai bao bien cuc bo \*/

int ketqua;

if (so1 > so2)

ketqua = so1;

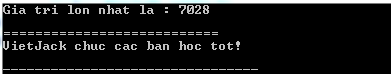
else

ketqua = so2;

return ketqua;

}

Tôi giữ giá trị hàm timGTLN() trong hàm main vào biến *kq*. Biên dịch và chạy chương trình C trên sẽ cho kết quả:



Tham số của hàm trong C

Một hàm sử dụng các danh sách tham số, nó phải khai báo các biến và chấp nhận giá trị các biến này. Các biến này được gọi là các **biến chính thức**.

Các biến chính thức giống các biến cục bộ khác bên trong hàm.

Khi bạn gọi hàm, có 2 cách để bạn truyền các giá trị vào cho hàm:

|  |  |
| --- | --- |
| **Kiểu gọi** | **Miêu tả** |
| [**Gọi bởi giá trị**](https://vietjack.com/lap_trinh_c/goi_ham_boi_gia_tri_trong_c.jsp) | Phương thức này sao chép giá trị thực sự của tham số vào trong tham số chính thức của một hàm. Trong trường hợp này, các thay đổi của bản thân các tham số bên trong hàm không ảnh hưởng tới các tham số. |
| [**Gọi bởi tham chiếu**](https://vietjack.com/lap_trinh_c/goi_ham_boi_tham_chieu_trong_c.jsp) | Phương thức này sao chép địa chỉ của tham số vào trong tham số chính thức. Bên trong hàm, địa chỉ được dùng để truy cập tham số thực sự được sử dụng khi gọi hàm. Có nghĩa là các thay đổi tới tham số làm tham số thay đổi. |

Theo mặc định, C sử dụng **gọi bởi giá trị** để truyền các tham số. Nhìn chung, code đó trong một hàm không thể thay đổi các tham số được dùng để gọi hàm đó và trong ví dụ trên, khi gọi hàm timGTLN() là dùng phương thức tương tự.

Quy tắc phạm vi trong C

[Trang trước](https://vietjack.com/lap_trinh_c/ham_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/mang_trong_c.jsp)

Một phạm vi trong bất kỳ chương trình nào là một khu vực trong chương trình nơi một biến được định nghĩa tồn tại và bên ngoài phạm vi đó thì biến không thể được truy cập. Có 3 nơi mà biến có thể được khai báo trong Ngôn ngữ C:

1. Bên trong một hàm hoặc một khối code được gọi là biến **cục bộ**.
2. Bên ngoài tất cả các hàm và được gọi là biến **toàn cục**.
3. Trong định nghĩa các hàm, các tham số được gọi là các tham số **chính thức - formal**.

Bây giờ chúng ta cùng tìm hiểu các biến **cục bộ** và biến **toàn cục** và các tham số **chính thức** là gì.

Biến cục bộ trong C

Biến **cục bộ** được khai báo bên trong một hàm hoặc khai báo bên trong một khối code. Chúng được sử dụng bởi các lệnh trong hàm hoặc khối code đó. Biến cục bộ **không** được sử dụng bên ngoài của hàm. Dưới đây là ví dụ về việc sử dụng biến cục bộ. Tại đây biến a,b và c được sử dụng trong hàm main():

#include <stdio.h>

int main ()

{

/\* Khai bao bien cuc bo \*/

int a, b;

int c;

/\* khoi tao thuc su \*/

a = 10;

b = 20;

c = a + b;

printf ("Gia tri cua a = %d, b = %d va c = %d\n", a, b, c);

return 0;

}

Biến toàn cục trong C

Biến **toàn cục** được định nghĩa bên ngoài một hàm, thường là phần đầu của chương trình. Biến toàn cục có thể chứa các giá trị trong thời gian chương trình chạy và có thể được truy cập bởi bất kì hàm nào định nghĩa trong chương trình.

Một biến toàn cục có thể được truy cập bởi bất kì hàm nào. Điều đó nghĩa là biến toàn cục được sử dụng suốt chương trình sau khi nó khai báo. Dưới đây là ví dụ minh họa cho biến cục bộ và toàn cục:

#include <stdio.h>

/\* Khai bao bien toan cuc \*/

int g;

int main ()

{

/\* Khai bao bien cuc bo \*/

int a, b;

/\* khoi tao bien thuc su \*/

a = 10;

b = 20;

g = a + b;

printf ("Gia tri cua a = %d, b = %d va g = %d\n", a, b, g);

return 0;

}

Một chương trình có thể có biến toàn cục và cục bộ trùng tên. Trong trường hợp đó biến cục bộ bên trong hàm sẽ được ưu tiên sử dụng. Dưới đây là ví dụ:

#include <stdio.h>

/\* Khai bao bien toan cuc \*/

int g = 20;

int main ()

{

/\* khai bao bien cuc bo \*/

int g = 10;

printf ("Gia tri cua g = %d\n", g);

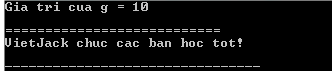
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

Khi chương trình C trên được biên dịch và chạy, kết quả sau sẽ in ra:



Tham số chính thức trong C

**Tham số hàm, tham số chính thức** được coi như biến **local** bên trong hàm đó và thường có giá trị ưu tiên hơn biến toàn cục. Dưới đây là ví dụ:

#include <stdio.h>

/\* khai bao bien toan cuc \*/

int a = 20;

/\* khai bao ham \*/

int hamtinhtong(int a, int b);

int main ()

{

/\* khai bao bien cuc bo trong ham main \*/

int a = 15;

int b = 25;

int c = 0;

printf ("Gia tri cua a trong ham main() = %d\n", a);

c = hamtinhtong( a, b);

printf ("Gia tri cua c trong ham main() = %d\n", c);

printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

/\* ham de cong hai so nguyen \*/

int hamtinhtong(int a, int b)

{

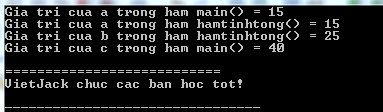
printf ("Gia tri cua a trong ham hamtinhtong() = %d\n", a);

printf ("Gia tri cua b trong ham hamtinhtong() = %d\n", b);

return a + b;

}

Khi chương trình C trên được biên dịch và chạy, kết quả sau sẽ in ra:



Khởi tạo biến toàn cục và biến cục bộ

Khi biến toàn cục và biến cục bộ được định nghĩa, nó không được khởi tạo bởi chương trình, nó phải khởi tạo bởi chính bản thân bạn. Biến toàn cục thường khởi tạo tự động bởi chương trình khi bạn định nghĩa chúng như dưới đây:

|  |  |
| --- | --- |
| **Kiểu dữ liệu** | **Giá trị khởi tạo mặc định** |
| int | 0 |
| char | '\0' |
| float | 0 |
| double | 0 |
| pointer | NULL |

Thường trong thực tế lập trình bạn nên khởi tạo các giá trị biến một cách chính xác, nếu không chương trình của bạn có thể gây ra những kết quả không mong muốn .

# Mảng trong Ngôn ngữ C

[Trang trước](https://vietjack.com/lap_trinh_c/bien_toan_cuc_va_bien_cuc_bo_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/con_tro_trong_c.jsp)

Ngôn ngữ lập trình C cung cấp cấu trúc dữ liệu gọi là **mảng**, được lưu trữ trong một tập hợp các dữ liệu cùng kiểu với độ dài cố định. Một mảng được sử dụng để lưu trữ tập hợp dữ liệu, nhưng nó rất hữu dụng nếu bạn nghĩ về một mảng các biến với cùng một kiểu.

Thay vì khai báo biến một cách rời rạc, như biến number0, number1,… và number99, bạn có thể khai báo một mảng các giá trị như numbers[0], numbers[1] và … numbers[99] để biểu diễn các giá trị riêng biệt. Một thành phần cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liền kề nhau. Địa chỉ thấp nhất tương ứng với thành phần đầu tiền và địa chỉ cao nhất tương ứng với thành phần cuối cùng của mảng.



## Khai báo mảng trong C

Để khai báo một mảng trong ngôn ngữ C, chương trình xác định kiểu của biến và số lượng các phần tử được yêu cầu bởi biến đó như sau:

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

Đây là **mảng một chiều**. **Kich\_co\_mang** phải là một số nguyên lớn hơn 0 và **Kieu** phải hợp lệ trong ngôn ngữ C. Ví dụ, khai báo một mảng 10 phần tử gọi là *sohangban* với kiểu int, sử dụng câu lệnh sau đây:

int sohangban[10];

Bây giờ *sohangban* là một biến mảng có thể đủ chỗ chứa *10* phần tử int.

## Khởi tạo mảng trong C

Bạn có thể khởi tạo mảng trong C hoặc từng phần tử một hoặc sử dụng một câu lệnh như dưới đây:

int sohangban[5] = {34, 56, 23, 124, 67};

Số lượng các giá trị trong dấu ngoặc kép {} không được lớn hơn số lượng phần tử khai báo trong dấu ngoặc vuông [].

Nếu bạn bỏ sót kích cỡ mảng thì mảng đó đủ lớn để giữ các giá trị được khởi tạo:

int sohangban[] = {34, 56, 23, 124, 67};

Bạn sẽ tạo chính xác một chuỗi có giá trị giống hệt chuỗi bên trên bằng cách gán từng phần tử một. Dưới đây là một ví dụ khi gán giá trị cho một phần tử của mảng:

sohangban[4] = 67;

Câu lệnh bên trên gán giá trị thứ 5 của mảng giá trị 67. Tất cả các mảng đều có chỉ số (index) đầu tiên bằng 0, đây được gọi là chỉ số cơ bản và phần tử cuối cùng của mảng có chỉ số bằng độ lớn của mảng trừ đi 1. Dưới đây là cách biểu diễn hình họa cho chuỗi khai báo bên trên thông qua chỉ số:



## Truy cập các phần tử mảng trong C

Một mảng được truy cập bởi cách đánh chỉ số trong tên của mảng. Dưới đây là một cách truy cập một giá trị của mảng:

int luonghangban = sohangban[9];

Câu lệnh trên lấy phần tử thứ 10 của mảng và gán giá trị này cho biến *luonghangban*. Dưới đây là một ví dụ về việc sử dụng với tất cả mô tả bên trên:

#include <stdio.h>

int main ()

{

int n[ 10 ]; /\* mang n gom 10 so nguyen \*/

int i,j;

/\* khoi tao cac phan tu trong mang ve gia tri 0 \*/

for ( i = 0; i < 10; i++ )

{

n[ i ] = i + 100; /\* Thiet lap phan tu tai vi tri i thanh i + 100 \*/

}

/\* hien thi gia tri cac phan tu trong mang \*/

for (j = 0; j < 10; j++ )

{

printf("Phan tu [%d] = %d\n", j, n[j] );

}

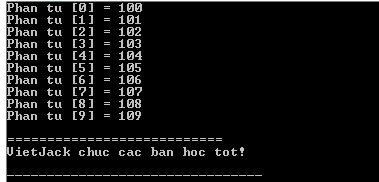
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



## Chi tiết về mảng trong C

Mảng là một phần rất quan trọng trong ngôn ngữ C. Dưới đây là những định nghĩa quan trọng liên quan đến một mảng cụ thể mà được trình bày rõ ràng hơn cho các lập trình viên C:

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Mảng đa chiều trong C**](https://vietjack.com/lap_trinh_c/mang_da_chieu_trong_c.jsp) | C hỗ trợ các mảng đa chiều. Mẫu đơn giản nhất của mảng này là mảng hai chiều. |
| [**Truyền mảng cho hàm như là tham số trong C**](https://vietjack.com/lap_trinh_c/truyen_mang_toi_ham_trong_c.jsp) | Bạn có thể truyền tới hàm một điểm trỏ chỉ tới một mảng bởi xác định tên mảng chứ không phải là một chỉ số. |
| [**Trả về mảng từ một hàm trong C**](https://vietjack.com/lap_trinh_c/tra_ve_mang_tu_ham_trong_c.jsp) | C cho phép một hàm có thể trả về một mảng. |
| [**Trỏ tới một mảng trong C**](https://vietjack.com/lap_trinh_c/con_tro_tro_toi_mang_trong_c.jsp) | Bạn có thể trỏ tới phần tử đầu tiên của mảng một cách đơn giản chỉ bởi xác định tên mảng đó, chứ không phải một chỉ số. |

# Con trỏ trong C

[Trang trước](https://vietjack.com/lap_trinh_c/mang_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/chuoi_trong_c.jsp)

**Con trỏ - Pointer** trong ngôn ngữ C rất dễ học. Một vài tác vụ trong ngôn ngữ C được thực hiện dễ dàng hơn nhờ con trỏ, và những tác vụ khác trở nên linh hoạt hơn, như trong việc cấp phát bộ nhớ, không thể thực hiện mà không dùng con trỏ. Do đó rất cần thiết phải nắm vững con trỏ khi trở thành một lập trình viên C hoàn thiện. Bây giờ hãy bắt đầu bằng những bước đơn giản nhất.

Như bạn biết, mỗi biến trong một vùng nhớ nhất định và mỗi vùng nhớ này có địa chỉ có nó được định nghĩa để dễ dàng trong việc truy cập sử dụng toán tử (&), tương ứng với địa chỉ của nó trong bộ nhớ. Xem xét ví dụ dưới đây, sẽ in ra địa chỉ của biến được định nghĩa:

#include <stdio.h>

int main ()

{

int bien1;

char bien2[25];

printf("Dia chi cua bien1 la: %x\n", &bien1 );

printf("Dia chi cua bien2 la: %x\n", &bien2 );

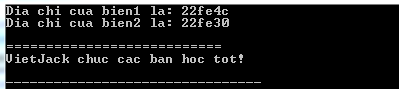
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Do đó bạn hiểu thế nào là địa chỉ bộ nhớ và cách truy cập nó, đó là phần định nghĩa cơ bản. Bây giờ hãy xem thế nào là một con trỏ.

## Con trỏ là gì?

Một **con trỏ - pointer** là một biến mà trong đó giá trị của nó là địa chỉ của biến khác. Ví dụ như địa chỉ của vùng nhớ. Giống như các biến và hằng số, bạn phải khai báo con trỏ trước khi bạn có thể sử dụng nó để lưu trữ bất kì địa chỉ của biến nào. Dạng tổng quát của việc khai báo con trỏ như sau:

kieu\_du\_lieu \*ten\_bien;

Ở đây, **kieu\_du\_lieu** là kiểu dữ liệu cơ bản con trỏ, nó là kiểu hợp lệ trong ngôn ngữ C và **var-ten\_bien** là tên giá trị của con trỏ. Phần ký tự \* sử dụng trong khai báo con trỏ giống như việc bạn sử dụng cho phép nhân. Mặc dù vậy, trong khai báo này, ký tự \* được thiết kế để sử dụng các biến của con trỏ. Dưới đây là một số cách khai báo hợp lệ của con trỏ:

int \*contro; /\* con tro tro toi mot so nguyen \*/

double \*phithuebao; /\* con tro tro toi mot so double \*/

float \*hocphi; /\* con tro tro toi mot so float \*/

char \*ho, \*ten; /\* con tro tro toi mot ky tu \*/

Kiểu dữ liệu thực sự của giá trị của tất cả các con trỏ, có thể là số nguyên, float, ký tự, hoặc kiểu khác như một số thập lục phân dài - Long hexa biểu diễn một địa chỉ bộ nhớ. Điểm khác nhau duy nhất của các con trỏ của các kiểu dữ liệu khác nhau là kiểu dữ liệu của biến hoặc hằng số mà con trỏ chỉ tới.

## Cách sử dụng con trỏ trong C ?

Có một vài phép toán quan trọng, sẽ giúp chúng ta làm việc với con trỏ một cách thường xuyên: **a)** chúng ta định nghĩa biến con trỏ, **b)** gán địa chỉ của biến đến một con trở và **c)** cuối cùng truy cập các giá trị biến địa chỉ trong biến con trỏ. Điều này được thực hiện bởi toán tử **\*** trả về giá trị các các biến chứa trong địa chỉ được xác định bởi toán tử này. Dưới đây là các sử dụng những phép toán trên:

#include <stdio.h>

int main ()

{

int bien = 20; /\* phan khai bao bien thuc su \*/

int \*contro; /\* phan khai bao bien con tro \*/

contro = &bien; /\* luu tru dia chi cua bien trong con tro \*/

printf("Dia chi cua bien la: %x\n", &bien );

/\* dia chi duoc luu tru trong bien con tro \*/

printf("Dia chi duoc luu tru trong bien contro la: %x\n", contro );

/\* Truy cap gia tri boi su dung con tro \*/

printf("Gia tri cua bien \*contro la: %d\n", \*contro );

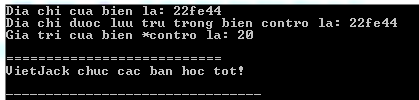
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



## Con trỏ NULL trong C

Có một cách thực hành tốt khi chúng ta gán giá trị NULL cho biến con trở trong trường hợp bạn không có địa chỉ chính xác để được gán. Điều này thường xảy ra trong quá trình khai báo. Một con trỏ được gán giá trị NULL được gọi là con trỏ **null**.

Con trỏ null là một hằng số với giá trị 0 được định nghĩa trong một vài thư viện chuẩn. Xem chương trình dưới đây:

#include <stdio.h>

int main ()

{

int \*contro = NULL;

printf("Gia tri cua contro la: %x\n", contro );

return 0;

}

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

Gia tri cua contro la: 0

Trong hầu hết các hệ thống, chương trình không cho phép truy cập và ô nhớ có địa chỉ 0 bởi vì bộ nhớ này dùng để dự trữ cho hệ điều hành. Nếu con trỏ tới giá trị null, nó được coi là không trỏ tới đâu cả.

Để kiểm tra có phải là con trỏ null hay không bạn có thể sử dụng lệnh if như sau:

if(contro) /\* la true neu contro khong phai la null \*/

if(!contro) /\* la true neu contro la null \*/

## Chi tiết về con trỏ trong C

Con trỏ có nhiều nhưng dễ dàng trong việc định nghĩa và rất quan trọng trong lập trình ngôn ngữ C. Dưới đây là những định nghĩa quan trọng mà rõ ràng về con trỏ trong ngôn ngữ lập trình C:

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Con trỏ số học trong C**](https://vietjack.com/lap_trinh_c/con_tro_so_hoc_trong_c.jsp) | Có 4 toán tử đại số mà có thể được sử dụng trên các con trỏ là: ++, --, +, - |
| [**Mảng các con trỏ trong C**](https://vietjack.com/lap_trinh_c/mang_con_tro_trong_c.jsp) | Bạn có thể định nghĩa các mảng để giữ các con trỏ. |
| [**Con trỏ trỏ tới con trỏ trong C**](https://vietjack.com/lap_trinh_c/con_tro_tro_toi_con_tro_trong_c.jsp) | C cho phép bạn trỏ tới một con trỏ ... |
| [**Truyền các con trỏ tới hàm trong C**](https://vietjack.com/lap_trinh_c/truyen_con_tro_toi_ham_trong_c.jsp) | Truyền một tham số bởi tham chiếu hoặc địa chỉ: cả hai cho các tham số được truyền khả năng có thể được thay đổi trong hàm gọi bởi hàm được gọi. |
| [**Trả về con trỏ từ hàm trong C**](https://vietjack.com/lap_trinh_c/tra_ve_con_tro_tu_ham_trong_c.jsp) | C cho phép một hàm trả về một con trỏ tới biến cục bộ, biến tĩnh và cũng như bộ nhớ được cấp phát động. |

# Chuỗi trong C

[Trang trước](https://vietjack.com/lap_trinh_c/con_tro_trong_c.jsp)

[Trang sau](https://vietjack.com/lap_trinh_c/cau_truc_struct_trong_c.jsp)

Chuỗi trong ngôn ngữ lập trình C thực chất là mảng một chiều của các ký tự mà kết thúc bởi một ký tự **null** '\0'.

Phần khai báo và khởi tạo dưới đây tạo ra một chuỗi bao gồm một từ "Hello". Để giữ các giá trị null tại cuối của mảng, cỡ của mảng các ký tự bao gồm một chuỗi phải nhiều hơn số lượng các ký tự trong từ khóa "Hello".

char loichao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

Nếu bạn theo quy tắc khởi tạo các chuỗi, bạn có thể viết lệnh như sau:

char loichao[] = "Hello";

Dưới đây là phần biểu diễn ô nhớ cho đoạn chuỗi trên trong ngôn ngữ C/C++:



Thực tế, bạn không đặt ký tự **null** tại vị trí cuối cùng của biến hằng số. Bộ biên dịch C tự động thêm '\0' tại ví trí cuối cùng của chuỗi khi nó khởi tạo chuỗi. Cùng thử ví dụ in ra chuỗi sau đây:

#include <stdio.h>

int main ()

{

char loichao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

printf("Khi gap nhau, ban chao: %s\n", loichao );

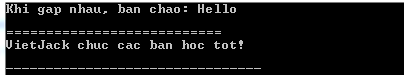
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Ngôn ngữ C hỗ trợ một dãy rộng rãi các hàm để thao tác các chuỗi kết thúc là null:

|  |  |
| --- | --- |
| **STT** | **Hàm & Mục đích** |
| 1 | **strcpy(s1, s2);**  Sao chép chuỗi s2 cho chuỗi s1. |
| 2 | **strcat(s1, s2);**  Nối chuỗi s2 vào cuối chuỗi s1. |
| 3 | **strlen(s1);**  Trả về độ dài của chuỗi s1. |
| 4 | **strcmp(s1, s2);**  Trả về 0 nếu s1 và s2 là như nhau; nhỏ hơn 0 nếu s1<s2; lớn hơn 0 nếu s1>s2. |
| 5 | **strchr(s1, ch);**  Trả về con trỏ tới vị trí đầu tiên của ch trong s1. |
| 6 | **strstr(s1, s2);**  Trả về con trỏ tới vị trí đầu tiên của chuỗi s2 trong chuỗi s1. |

Dưới đây là ví dụ cho việc sử dụng một vài hàm bên trên:

#include <stdio.h>

#include <string.h> /\* thu vien cho cac ham xu ly chuoi\*/

int main ()

{

char chuoi1[12] = "Hello";

char chuoi2[12] = "Vietjack";

char chuoi3[12];

int dodai ;

/\* sao chep chuoi1 vao trong chuoi3 \*/

strcpy(chuoi3, chuoi1);

printf("Ban su dung ham strcpy( chuoi3, chuoi1) de sao chep: %s\n", chuoi3 );

/\* noi hai chuoi: chuoi1 va chuoi2 \*/

strcat( chuoi1, chuoi2);

printf("Ban su dung ham strcat( chuoi1, chuoi2) de noi chuoi: %s\n", chuoi1 );

/\* tinh do dai cua chuoi1 sau khi noi chuoi \*/

dodai = strlen(chuoi1);

printf("Ban su dung ham strlen(chuoi1) de tinh do dai: %d\n", dodai );

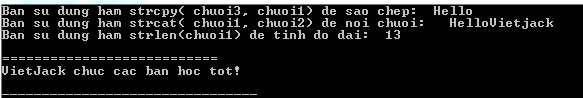
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



Bạn có thể tìm thấy một danh sách đầy đủ các hàm liên quan tới chuỗi trong Thư viện tiêu chuẩn C.