# Lập trình cơ bản C/C++

### Giới thiệu C++

C++ là một ngôn ngữ lập trình kiểu tĩnh,dữ liệu trừ tượng, phân biệt kiểu chữ thường chữ hoa mà hỗ trợ lập trình hướng đối tượng, lập trình thủ tục.

C++ được coi như là ngôn ngữ **bậc trung (middle-level)**, khi nó kết hợp các đặc điểm và tính năng của ngôn ngữ bậc cao và bậc thấp.

C++ được phát triển bởi Bjarne Stroustrup năm 1979 tại Bell Labs ở Murray Hill, New Jersey, như là một bản nâng cao của ngôn ngữ C và với tên gọi đầu tiên là "C với các Lớp", nhưng sau đó được đổi tên thành C++ vào năm 1983.

C++ là một Superset của C, và bất kỳ chương trình C nào cũng là một chương trình C++.

**Ghi chú:** Khi nói một ngôn ngữ lập trình là kiểu tĩnh khi việc kiểm tra kiểu được thực hiện trong thời gian biên dịch (compile-time), chứ không phải là run-time.

1. Lập trình hướng đối tượng

C++ hỗ trợ đầy đủ lập trình hướng đối tượng, bao gồm 4 tính năng trụ cột của lập trình hướng đối tượng là:

* Tính bao đóng (Encapsulation)
* Ẩn dữ liệu (Data hiding)
* Tính kế thừa (Inheritance)
* Tính đa hình (Polymorphism)

1. Thư việc chuẩn C++ (C++ Standard Library)

C++ chuẩn gồm 3 phần quan trọng:

* Core Language cung cấp tất cả các khối bao gồm biến, kiểu dữ liệu (data type) và literals, …
* Thư viện chuẩn C++ (C++ Standard Library) cung cấp tập hợp hàm đa dạng để thao tác file, string, …
* Standard Template Library (STL) cung cấp tập hợp phương thức đa dạng để thao tác cấu trúc dữ liệu, …

## ANSI Standard

ANSI Standard bảo đảm rằng C++ là portable, nghĩa là: code bạn viết cho compiler của Microsoft sẽ được biên dịch mà không xảy ra lỗi khi sử dụng một compiler trên Mac, UNIX, và Windows box hoặc một Alpha.

ANSI Standard đã tồn tại trong thời gian dài, và tất cả C++ compiler đều hỗ trợ ANSI Standard.

## Học tập C++ như thế nào?

**Điều quan trọng nhất khi học tập C++ là trọng tâm vào các khái niệm.**

Mục đích của việc học một ngôn ngữ lập trình là trở thành một lập trình viên giỏi hơn, tức là để thiết kế và thực thi các hệ thống mới hiệu quả hơn, hoặc sửa đổi chương trình cũ.

C++ hỗ trợ các phong cách lập trình đa dạng. Bạn có thể viết trong Fortran, C, Smalltalk, … trong bất kỳ ngôn ngữ nào. Mỗi style có thể đạt được mục tiêu của nó một cách hiệu quả trong khi duy trì runtime và space hiệu quả.

## Miền sử dụng của C++

C++ được sử dụng bởi hàng nghìn lập trình viên trong mỗi lĩnh vực ứng dụng cơ bản.

C++ đang được sử dụng phổ biến để viết device driver và các phần mềm khác mà dựa vào thao tác trực tiếp của phần cứng dưới ràng buộc về realtime.

C++ được sử dụng rộng rãi để đào tạo, nghiên cứu.

Bất cứ hai đã sử dụng Apple Macintosh hoặc một PC chạy trên Windows đã là gián tiếp sử dụng C++, bởi vì User Interface của các hệ thống này được viết bằng C++.

### Cài đặt IDE

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**.

### Cú pháp C++ cơ bản

Khi chúng ta xem xét một chương trình C++, nó có thể được định nghĩa như là một tập hợp của các đối tượng, mà giao tiếp thông qua việc triệu hồi các phương thức của mỗi đối tượng đó. Dưới đây, chúng tôi miêu tả ngắn gọn ý nghĩa của class (lớp), object (đối tượng), method (phương thức) và các biến đối tượng:

* **Đối tượng -** Đối tượng có các trạng thái và hành vi. Ví dụ: một đối tượng dog có các trạng thái là color, name, breed, và các hành vi là wagging, barking, eating. Một đối tượng là một minh họa của một lớp.
* **Lớp -** Một lớp có thể được định nghĩa như là một template/blueprint, mà miêu tả hành vi/trạng thái mà đối tượng hỗ trợ.
* **Phương thức -** Về cơ bản, một phương thức là một hành vi. Một lớp có thể chứa nhiều phương thức. Phương thức là nơi tính logic được viết, dữ liệu được thao tác và tất cả action được thực thi.
* **Biến instance -** Mỗi đối tượng có tập hợp biến đối tượng duy nhất của nó. Trạng thái của một đối tượng được tạo ra bởi các giá trị được gán cho các biến đối tượng của nó.

**Cấu trúc chương trình C++:**

Bạn theo dõi một đoạn code đơn giản sẽ in *Hello World*.

#include <iostream>

using namespace std;

// Ham main() la noi su thuc thi chuong trinh bat dau

int main()

{

cout << "Hello World"; // In dong chu Hello World

return 0;

}

**Chương trình trên có các phần sau:**

* Ngôn ngữ C++ định nghĩa một số header, mà chứa tong tin cần thiết và hữu ích cho chương trình của bạn. Với chương trình này, header là **<iostream>** là cần thiết.
* Dòng **using namespace std;** nói cho compiler sử dụng std namespace. Namespce là phần bổ sung gần đây cho C++.
* Dòng tiếp theo **Ham main() la noi su thuc thi chuong trinh bat dau** là một comment đơn dòng trong C++. Các comment đơn dòng bắt đầu với // và kết thúc ở cuối dòng.
* Dòng **int main()** là hàm main, tại đây việc thực thi chương trình bắt đầu.
* Dòng tiếp theo **cout << "Hello World";** để in dòng chữ "Hello World" trên màn hình.
* Dòng tiếp theo **return 0;** kết thúc hàm main() và làm nó trả về giá trị 0 tới tiến trình đang gọi.

Biên dịch và thực thi chương trình C++

**1.** Dưới đây là cách lưu file, biên dịch và chạy chương trình với command prompt. Bạn theo các bước sau:

* Mở một text editor và thêm đoạn code trên.
* Lưu file với tên: hello.cpp.
* Mở một dòng nhắc lệnh (command prompt) và tới thư mục nơi bạn lưu file đó.
* Soạn 'g++ hello.cpp ' và nhấn Enter để biên dịch code trên. Nếu không có lỗi xảy ra trong code của bạn thì dòng nhắc lệnh sẽ đưa bạn tới dòng tiếp theo và tạo a.out file có thể thực thi.
* Bây giờ, bạn soạn 'a.out' để chạy chương trình.
* Bạn sẽ thấy 'Hello World' được in trên cửa sổ.

$ g++ hello.cpp

$ ./a.out

Hello World

Bạn chắc chắn rằng g++ là trong path của bạn và bạn đang chạy nó trong thư mục mà chứa hello.cpp file.

**2.** Đó là cách biên dịch và thực thi chương trình tại command prompt. Nếu bạn đang sử dụng **Dev-C++, hoặc Microsoft Visual Studio, hoặc Turbo C++**, bạn có thể sao chép ví dụ trên, sau đó:

Mở Dev-C++ chẳng hạn, chọn **File -> New Source**, sau đó dán ví dụ vào sourve file này và chọn **Execute** tab, chọn **Complile & Run** để thực thi chương trình. Bạn cũng có thể tạo một Project mới trong tùy chọn New để đặt tên và lưu cho ví dụ vừa thực hiện.

Dấu chấm phảy và khối (block) trong C++

Trong C++, dấu chấm phảy là ký tự kết thúc lệnh (statement terminator). Nghĩa là, mỗi lệnh đơn phải kết thúc bởi một dấu chấm phảy. Nó chỉ dẫn sự kết thúc của một thực thể logic.

Dưới đây là ví dụ về 3 lệnh khác nhau:

x = y;

y = y+1;

add(x, y);

Một khối (block) là một tập hợp các lệnh được kết nối một cách logic, mà được bao quanh bởi các dấu ngoặc móc mở và đóng. Ví dụ:

{

cout << "VietJack xin chao cac ban!"; // in dong chu VietJack xin chao cac ban!

return 0;

}

Hai dạng sau là tương đương, C++ không quan tâm bạn đặt bao nhiêu lệnh trên một dòng. Ví dụ:

x = y;

y = y+1;

add(x, y);

Là giống với:

x = y; y = y+1; add(x, y);

Định danh (Identifier) trong C++

Một Định danh (Identifier) trong C++ là một tên được sử dụng để nhận diện một biến, hàm, lớp, module, hoặc bất kỳ user-defined item nào (người dùng tự định nghĩa). Một Định danh (Identifier) bắt đầu với một chữ cái từ A tới Z hoặc từ a tới z hoặc một dấu gạch dưới (\_) được theo sau bởi 0 hoặc nhiều chữ cái, dấu gạch dưới và chữ số (từ 0 tới 9).

C++ không cho phép các ký tự như @, $ và % bên trong các Identifier. C++ là ngôn ngữ lập trình phân biệt kiểu chữ. Vì thế, **Manpower** và **manpower** là hai Identifier khác nhau trong C++.

Dưới đây là một số ví dụ về Identifier (Định danh) thích hợp:

hoang nam abc sinh\_vien a\_123

caogia50 \_nhanvien j a23b9 vietJack

Từ khóa trong C++

Bảng dưới liệt kê các từ được dự trữ (dành riêng) trong C++. Những từ này không thể được sử dụng như là constant hoặc biến hoặc bất kỳ tên Identifier (Định danh) nào.

|  |  |  |  |
| --- | --- | --- | --- |
| asm | else | new | this |
| auto | enum | operator | throw |
| bool | explicit | private | true |
| break | export | protected | try |
| case | extern | public | typedef |
| catch | false | register | typeid |
| char | float | reinterpret\_cast | typename |
| class | for | return | union |
| const | friend | short | unsigned |
| const\_cast | goto | signed | using |
| continue | if | sizeof | virtual |
| default | inline | static | void |
| delete | int | static\_cast | volatile |
| do | long | struct | wchar\_t |
| double | mutable | switch | while |
| dynamic\_cast | namespace | template |  |

Khoảng trắng (Whitespace) là khái niệm được sử dụng trong C++ để miêu tả blank, tab, ký tự newline (dòng mới), và comment. Khoảng trắng (Whitespace) phân biệt các phần của lệnh và giúp compiler nhận diện vị trí một phần tử trong một lệnh, ví dụ như int, vị trí phần kết thúc và vị trí phần tử tiếp theo bắt đầu. Do đó, trong lệnh sau:

int diemthi;

Phải có ít nhất một Whitespace (thường là khoảng trống) giữa int và age để compiler có thể phân biệt chúng. Trong lệnh:

tongLuong = luongCoBan + phuCap; // tinh tong luong

Các ký tự khoảng trắng (Whitespace) giữa luongCoBan và =, hoặc giữa = và phuCap là không cần thiết; tuy nhiên, để giúp cho code của bạn dễ đọc hơn, bạn có thể thêm chúng vào.

### Comment trong C/C++

Comment của chương trình là các lời diễn giải, mà bạn có thể bao trong C/C++ code, và giúp cho bất kỳ ai đọc source code dễ dàng hơn. Tất cả ngôn ngữ lập trình đều cho phép một số mẫu comment nào đó.

C++ hỗ trợ các comment đơn dòng và đa dòng. Tất cả ký tự có trong comment được bỏ qua bởi C/C++ compiler.

Comment trong C/C++ bắt đầu với /\* và kết thúc với \*/. Ví dụ:

/\* Day la mot comment don dong \*/

/\* C/C++ cung ho tro cac comment

\* ma co nhieu dong

\*/

Một comment cũng có thể bắt đầu với //, kéo dài tới phần cuối của dòng. Ví dụ:

cout << "Hoc C/C++ co ban va nang cao"; // In dong chu Hoc C/C++ co ban va nang cao

Khi code trên được biên dịch, nó sẽ bỏ qua **// In dong chu Hello World** và cuối cùng cho kết quả sau:

Hello World

Bên trong một comment dạng /\* và \*/, các ký tự // không có ý nghĩa đặc biệt gì. Bên trong một comment dạng //, các ký tự /\* và \*/ không có ý nghĩa đặc biệt gì. Vì thế, bạn có thể "lồng" bất kỳ dạng comment nào bên trong dạng khác. Ví dụ:

/\* Comment ve dong lenh in dong chu Hello World:

cout << "VietJack xin chao cac ban!"; // in dong chu VietJack xin chao cac ban!

\*/

### Kiểu dữ liệu

**Kiểu dữ liệu trong C/C++**Trong khi làm việc với bất kỳ ngôn ngữ lập trình nào, bạn cần sử dụng các kiểu biến đa dạng để lưu giữ thông tin. Các biến, không gì khác ngoài các vị trí bộ nhớ được dành riêng để lưu giá trị. Nghĩa là, khi bạn tạo một biến, bạn dành riêng một số không gian trong bộ nhớ cho biến đó.

Tên tiếng Anh là Primitive Type, còn có thể gọi là kiểu dữ liệu gốc, kiểu dữ liệu có sẵn trong C/C++. Bên cạnh các kiểu dữ liệu gốc này, C/C++ cũng cung cấp các kiểu dữ liệu user-defined. Bảng sau hiển thị kiểu biến, lượng bộ nhớ nó dùng để lưu giá trị trong bộ nhớ, và giá trị lớn nhất và nhỏ nhất có thể được lưu giữ với các kiểu biến đó:

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Độ rộng bit** | **Dãy giá trị** |
| char | 1 byte | -127 tới 127 hoặc 0 tới 255 |
| unsigned char | 1 byte | 0 tới 255 |
| signed char | 1 byte | -127 tới 127 |
| int | 4 byte | -2147483648 tới 2147483647 |
| unsigned int | 4 byte | 0 tới 4294967295 |
| signed int | 4 byte | -2147483648 tới 2147483647 |
| short int | 2 byte | -32768 tới 32767 |
| unsigned short int | Range | 0 tới 65,535 |
| signed short int | Range | -32768 tới 32767 |
| long int | 4 byte | -2,147,483,647 tới 2,147,483,647 |
| signed long int | 4 byte | Tương tự như long int |
| unsigned long int | 4 byte | 0 tới 4,294,967,295 |
| float | 4 byte | +/- 3.4e +/- 38 (~7 chữ số) |
| double | 8 byte | +/- 1.7e +/- 308 (~15 chữ số) |
| long double | 8 byte | +/- 1.7e +/- 308 (~15 chữ số) |
| wchar\_t | 2 hoặc 4 byte | 1 wide character |

* Quảng cáo

**Khai báo typedef trong C/C++**

* Bạn có thể tạo một tên mới cho một kiểu dữ liệu đang tồn tại bởi sử dụng **typedef** trong C/C++. Cú pháp đơn giản sau để định nghĩa một kiểu dữ liệu mới bởi sử dụng typedef:
* typedef kieu\_du\_lieu ten\_moi;
* Ví dụ sau nói cho compiler rằng sothuc là tên khác của float:
* typedef float sothuc;
* Bây giờ, khai báo sau là hoàn toàn hợp lệ và sẽ tạo một biến số thực gọi là *vantoc*:
* sothuc vantoc;
* Quảng cáo

Kiểu liệt kê enum trong C/C++

Kiểu liệt kê enum khai báo một tên kiểu tùy ý và một tập hợp của 0 hoặc nhiều Identifier (Định danh) mà có thể được sử dụng như là các giá trị của kiểu đó. Mỗi Enumerator là một constant có kiểu là kiểu liệt kê (enumeration).

Để tạo một Enumeration, bạn sử dụng từ khóa **enum** trong C/C++. Form chung của kiểu liệt kê enum là:

enum ten\_cua\_enum { danh sach cac ten } danh\_sach\_bien;

Tại đây, ten\_cua\_enum là tên kiểu liệt kê. Danh sách tên được phân biệt bởi dấu phảy.

Ví dụ, code sau định nghĩa một tên kiểu liệt kê hàng hóa gọi là *hanghoa* và biến c là kiểu của hanghoa. Cuối cùng, c được gán giá trị nuocngot.

### Các kiểu biến

Một biến cung cấp nơi lưu giữ được đặt tên để chúng ta có thể thao tác. Mỗi biến trong C/C++ có một kiểu cụ thể, mà quyết định: kích cỡ và cách bố trí bộ nhớ của biến; dãy giá trị có thể được lưu giữ bên trong bộ nhớ đó; và tập hợp hoạt động có thể được áp dụng cho biến đó.

Tên biến có thể gồm các ký tự, các chữ số, dấu gạch dưới. Nó phải bắt đầu bởi hoặc một ký tự hoặc một dấu gạch dưới. Các ký tự chữ hoa và chữ thường là khác nhau bởi C/C++ là ngôn ngữ phân biệt kiểu chữ. Biến có thể trong các kiểu giá trị đa dạng như kiểu char, int, float, ...

## Định nghĩa biến trong C/C++

Định nghĩa biến trong C/C++ nghĩa là nói cho compiler nơi và lượng bộ nhớ cần tạo để lưu giữ biến đó. Một định nghĩa biến xác định một kiểu dữ liệu, và chứa danh sách của một hoặc nhiều biến có kiểu đó, như sau:

kieu\_gia\_tri danh\_sach\_bien;

Ở đây, **kieu\_gia\_tri** phải là một kiểu dữ liệu hợp lệ trong C/C++, gồm char, w\_char, int, float, double, bool hoặc bất kỳ đối tượng nào mà người dùng tự định nghĩa, … và **danh\_sach\_bien** có thể chứa một hoặc nhiều tên Identifier (Định danh) phân biệt nhau bởi dấu phảy. Sau đây là một số khai báo hợp lệ trong C/C++:

int j, q, k;

char v, viet;

float x, diemthi;

double luong;

Dòng **int j, q, k;** vừa khai báo và định nghĩa các biến j, q, k, mà chỉ dẫn compiler để tạo các biến với tên là j, q, k với kiểu int.

Các biến có thể được khởi tạo (được gán giá trị ban đầu) trong các khai báo. Initializer (phần khởi tạo) gồm một ký hiệu bằng được theo sau bởi một Constant Expression (biểu thức hằng số), như sau:

kieu\_gia\_tri ten\_bien = giaTri;

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 = 'vietjack'; // bien x co gia tri la 'vietjack'.

Với định nghĩa mà không có phần khởi tạo: các biến được khởi tạo với NULL (tất cả byte có giá trị 0); giá trị khởi tạo của tất cả biến khác không được định nghĩa.

**Khai báo biến trong C/C++**

Trong ví dụ sau, một biến được khai báo tại ở phần đầu chương trình, nhưng nó đã được định nghĩa ở bên trong hàm main.

#include <iostream>

using namespace std;

// Phan khai bao bien:

extern int a, b;

extern int c;

extern float f;

int main ()

{

// Phan dinh ngia bien:

int a, b;

int c;

float f;

// Phan khoi tao bien

a = 10;

b = 20;

c = a + b;

cout << c << endl ;

f = 70.0/3.0;

cout << f << endl ;

return 0;

}

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

30

23.3333

Giống khái niệm áp dụng trên khai báo hàm, bạn cung cấp tên hàm tại thời điểm khai báo và định nghĩa thực sự của hàm đó có thể được cung cấp ở bất cứ đâu. Ví dụ:

// phan khai bao ham

int func();

int main()

{

// phan goi ham

int i = func();

}

// phan dinh nghia ham

int func()

{

return 0;

}

### Phạm vi biến

Một scope (phạm vi) là một khu vực của chương trình nơi biến hoạt động, và nói chung có thể có 3 khu vực mà biến có thể được khai báo:

* Bên trong một hàm hoặc một khối, được gọi là biến cục bộ (local).
* Trong định nghĩa của các tham số hàm, được gọi là các tham số chính thức (formal).
* Bên ngoài của tất cả hàm, được gọi là biến toàn cục (global).

## Biến cục bộ trong C++

Các biến được khai báo bên trong một hàm hoặc khối là các biến cục bộ (local). Chúng chỉ có thể được sử dụng bởi các lệnh bên trong hàm hoặc khối code đó. Các biến cục bộ không được biết ở bên ngoài hàm đó (tức là chỉ được sử dụng bên trong hàm hoặc khối code đó). Dưới đây là ví dụ sử dụng các biến cục bộ:

#include <iostream>

using namespace std;

int main ()

{

// phan khai bao bien cuc bo:

int a, b;

int c;

// phan khoi tao bien

a = 10;

b = 20;

c = a + b;

cout << c;

return 0;

}

## Biến toàn cục trong C++

Biến toàn cục (global) trong C++ được định nghĩa bên ngoài các hàm, thường ở phần đầu chương trình. Các biến toàn cục giữ giá trị của nó trong suốt vòng đời chương trình của bạn.

Một biến toàn cục có thể được truy cập bởi bất kỳ hàm nào. Tức là, một biến toàn cục là có sẵn cho bạn sử dụng trong toàn bộ chương trình sau khi đã khai báo nó. Dưới đây là ví dụ sử dụng biến toàn cục và biến nội bộ trong C++:

#include <iostream>

using namespace std;

// phan khai bao bien toan cuc:

int g;

nt main ()

{

// phan khai bao bien cuc bo:

int a, b;

// phan khoi tao bien

a = 10;

b = 20;

g = a + b;

cout << g;

return 0;

}

Một chương trình có thể có các biến toàn cục và biến cục bộ cùng tên với nhau, nhưng trong một hàm thì giá trị của biến cục bộ sẽ được ưu tiên. Ví dụ:

#include <iostream>

using namespace std;

// phan khai bao bien toan cuc:

int g = 20;

int main ()

{

// phan khai bao bien cuc bo:

int g = 10;

cout << g;

return 0;

}

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

10

## Khởi tạo biến cục bộ và biến toàn cục bởi hệ thống trong C++

Khi một biến cục bộ được định nghĩa, nó không được khởi tạo bởi hệ thống, chính bạn phải khởi tạo nó. Các biến toàn cục được khởi tạo tự động bởi hệ thống khi bạn định nghĩa chúng, như sau:

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

Khởi tạo biến một cách chính xác là một sự thực hành tốt, nếu không, đôi khi chương trình sẽ cho kết quả không mong đợi.

### Hằng

Constant liên quan tới các giá trị cố định mà chương trình không thể thay đổi và chúng được gọi là **literals**.

Constant là một kiểu dữ liệu thay thế cho Literal, còn Literal thể hiện chính nó. Trong ví dụ: const PI = 3.14 thì Constant ở đây là PI, còn Literal là 3.14.

Constant có thể là bất kỳ kiểu dữ liệu cơ bản nào trong C/C++, và có thể được phân chia thành giá trị hằng số nguyên, hằng số thực, hằng ký tự, hằng chuỗi và Boolean literal (tạm dịch: hằng logic).

**Hằng số nguyên** có thể là decimal (cơ số 10), octal (cơ số 8) hay hexadecimal (cơ số 16). Giá trị có tiền tố (prefix) là 0 cho octal, là 0x hay 0X cho hexadecimal và không có gì cho decimal.

Một hằng số nguyên cũng có các hậu tố (suffix) U hay L thể hiện kiểu unsigned hay long. Hậu tố có thể là chữ hoa hoặc chữ thường và có thể trong bất kỳ thứ tự nào.

Ví dụ về các hằng số nguyên:

212 // Hop le

215u // Hop le

0xFeeL // Hop le

048 // Khong hop le: 8 khong phai la mot ky so trong he octal

032UU // Khong hop le: hau to (suffix) khong the bi lap

Dưới đây là các kiểu hằng số nguyên đa dạng:

85 // Hang so dang decimal

0213 // Hang so dang octal

0x4b // Hang so dang hexadecimal

30 // Hang so dang so nguyen (int)

30u // Hang so dang so nguyen khong dau (unsigned int)

30l // Hang so dang long

30ul // Hang so dang unsigned long

## Hằng số thực trong C/C++

Một hằng số thực bao gồm phần nguyên (integer part), dấu chấm thập phân (decimal point), phần lẻ (fraction part) và phần mũ (exponent part). Chúng ta có thể biểu diễn hằng số thực theo dạng decimal hay dạng mũ.

Khi thể hiện dạng decimal phải bao gồm dấu chấm thập phân, phần mũ hoặc cả hai. Khi thể hiện dạng mũ phải gồm phần nguyên, phần lẻ hoặc cả hai. Dạng mũ đi kèm với kí tự E hoặc e.

Xét các ví dụ sau:

3.14159 // Hop le

314159E-5L // Hop le

510E // Khong hop le: pham mu (exponent) con thieu

210f // Khong hop le: khong co phan thap pha decimal hoac phan mu exponent

.e55 // Khong hop le: thieu phan nguyen integer hoac phan fraction

## Boolean literal trong C/C++

Có hai kiểu Boolean literal và chúng là một phần của các từ khóa C/C++:

* Giá trị **true**
* Và giá trị **false**

## Sử dụng bộ tiền xử lý #define trong C/C++

Dưới đây là cú pháp để sử dụng tiền xử lý #define để định nghĩa một constant trong C/C++:

#define identifier value

Ví dụ sau giải thích cú pháp trên:

#include <iostream>

using namespace std;

#define CHIEUDAI 10

#define CHIEURONG 5

#define NEWLINE '\n'

int main()

{

int dientich;

dientich = CHIEUDAI \* CHIEURONG;

cout << dientich;

cout << NEWLINE;

return 0;

}

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

50

## Sử dụng từ khóa const trong C/C++

Bạn có thể sử dụng tiền tố **const** để định nghĩa hằng trong C/C++ với một kiểu cụ thể, như sau:

const kieu\_gia\_tri bien = giaTri;

Ví dụ sau giải thích chi tiết cú pháp trên:

#include <iostream>

using namespace std;

int main()

{

const int CHIEUDAI = 10;

const int CHIEURONG = 5;

const char NEWLINE = '\n';

int area;

dientich = CHIEUDAI \* CHIEURONG;

cout << dientich;

cout << NEWLINE;

return 0;

}

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

50

### Các kiểu Modifier

C++ cho phép các kiểu dữ liệu **char, int và double** có các Modifier đặt trước chúng. Một Modifier được sử dụng để thông báo ý nghĩa của kiểu cơ sở, giúp cho nó tăng sự chính xác hơn với sự cần thiết của các tình huống đa dạng.

Dưới đây là các Modifier trong C/C++:

* signed (có dấu)
* unsigned (không có dấu)
* long
* short

Các Modifier là: **signed, unsigned, long,** và **short** có thể được áp dụng cho kiểu integer. Ngoài ra, **signed** và **unsigned** có thể được áp dụng cho kiểu char, và **long** có thể áp dụng cho kiểu double.

Các Modifier là **signed** và **unsigned** cũng có thể được sử dụng như là tiền tố cho các Modifier là **long** hoặc **short** modifiers. Ví dụ: **unsigned long int**.

C++ cho phép kiểu khai báo tắt để khai báo các **unsigned, short,** or **long** integer. Bạn có thể chỉ đơn giản sử dụng từ **unsigned, short,** hoặc **long**, mà không cần int. Ví dụ sau minh họa hai khai báo là hợp lệ trong C/C++ để khai báo các

biến unsigned integer:

unsigned x;

unsigned int y;

Để phân biệt sự khác nhau giữa hai Modifier là signed integer và unsigned integer được thông dịch bởi C/C++, bạn nên chạy chương trình sau:

#include <iostream>

using namespace std;

/\* Chuong trinh nay chi ra diem khac nhau giua

\* cac so nguyen signed va unsigned.

\*/

int main()

{

short int i; // mot so nguyen signed short int

short unsigned int j; // mot so nguyen unsigned short int

j = 32769;

i = j;

cout << i << " " << j;

return 0;

}

Nó sẽ cho kết quả:

-32767 32769

Nếu bạn quay trở lại chương **Kiểu dữ liệu trong C/C++**, và đọc phần dãy giá trị của short int và unsigned short int, bạn sẽ nhận ra sự khác nhau khi chạy chương trình trên với j <= 32767 và với j >= 32767.

### Lớp lưu trữ

Lớp lưu trữ (Storage Class) định nghĩa phạm vi và vòng đời của biến và/hoặc các hàm bên trong một chương trình C/C++. Chúng thường đứng trước kiểu dữ liệu mà chúng tác động. Dưới đây là các lớp lưu trữ có thể được sử dụng trong C/C++:

* auto
* register
* static
* extern
* mutable

## Lớp lưu trữ auto trong C/C++

Lớp lưu trữ **auto** trong C/C++ là lớp lưu trữ mặc định cho tất cả biến cục bộ trong C/C++:

{

int diemthi;

auto int diemthi;

}

Ví dụ trên định nghĩa hai biến với cùng lớp lưu trữ, auto chỉ có thể được sử dụng bên trong các hàn, ví dụ: cho các biến nội bộ.

**Lớp lưu trữ register trong C/C++**

Lớp lưu trữ **register** trong C/C++ được sử dụng để định nghĩa các biến cục bộ mà nên được lưu giữ trong một thanh ghi thay vì RAM. Nghĩa là, biến có kích cỡ tối đa bằng với kích cỡ thanh ghi (thường là 1 từ) và không thể có toán tử một ngôi '&' được áp dụng tới nó (vì không có địa chỉ bộ nhớ).

{

register int hocphi;

}

Lớp lưu trữ register nên chỉ được dùng cho các biến yêu cầu truy cập nhanh như các biến đếm (counters). Cũng cần chú ý rằng, một biến định nghĩa với 'register' không có nghĩa là biến đó được lưu trữ trong thanh ghi. Tức là nó có thể được lưu trữ trong thanh ghi phụ thuộc vào phần cứng và giới hạn thực thi.

## Lớp lưu trữ static trong C/C++

Lớp lưu trữ **static** trong C/C++ nói với compiler để giữ một biến cục bộ tồn tại trong toàn bộ thời gian sống của chương trình thay vì tạo và hủy biến mỗi lần nó vào và ra khỏi phạm vi biến. Vì vậy, các biến có static cho phép nó duy trì giá trị giữa các lần gọi hàm.

#include <iostream>

// phan khai bao ham

void func(void);

static int biendem = 10; /\* Day la bien toan cuc \*/

main()

{

while(biendem--)

{

func();

}

return 0;

}

// Phan dinh nghia ham

void func( void )

{

static int i = 5; // Day la bien cuc bo dang static

i++;

std::cout << "i co gia tri la " << i ;

std::cout << " va biendem co gia tri la " << biendem << std::endl;

}

## Lớp lưu trữ extern trong C/C++

Lớp lưu trữ **extern** trong C/C++ được dùng để cung cấp một tham chiếu của một biến toàn cục được nhìn thấy bởi TẤT CẢ các file chương trình. Khi bạn sử dụng 'extern', biến không thể được khởi tạo, khi nó trỏ tới tên biến tại một vị trí lớp lưu trữ mà đã được định nghĩa trước đó.

Khi bạn có nhiều file và bạn định nghĩa một biến hay hàm toàn cục trong một file và cũng muốn dùng nó trong các file khác, thì **extern** được dùng trong file khác để cung cấp tham chiếu của biến hay hàm được định nghĩa. Cần nhớ rằng, **extern** dùng để khai báo một biến hay hàm toàn cục trong file khác.

### Toán tử

Một toán tử là một biểu tượng, mà nói cho compiler thực hiện các thao tác toán học và logic cụ thể. C++ cung cấp nhiều toán tử có sẵn, đó là:

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

[**Ví dụ toán tử số học**](https://vietjack.com/cplusplus/toan_tu_so_hoc_trong_cplusplus.jsp)

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Miêu tả** | **Ví dụ** |
| + | Cộng hai toán hạng | A + B kết quả là 30 |
| - | Trừ toán hạng thứ hai từ toán hạng đầu | A - B kết quả là -10 |
| \* | Nhân hai toán hạng | A \* B kết quả là 200 |
| / | Phép chia | B / A kết quả là 2 |
| % | Phép lấy số dư | B % A kết quả là 0 |
| ++ | [**Toán tử tăng (++)**](https://vietjack.com/cplusplus/toan_tu_tang_giam_trong_cplusplus.jsp), tăng giá trị toán hạng thêm một đơn vị | A++ kết quả là 11 |
| -- | [**Toán tử giảm (--)**](https://vietjack.com/cplusplus/toan_tu_tang_giam_trong_cplusplus.jsp), giảm giá trị toán hạng đi một đơn vị | A-- kết quả là 9 |

[**Ví dụ toán tử quan hệ**](https://vietjack.com/cplusplus/toan_tu_quan_he_trong_cplusplus.jsp)

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

[**Ví dụ toán tử logic**](https://vietjack.com/cplusplus/toan_tu_logic_trong_cplusplus.jsp)

|  |  |  |
| --- | --- | --- |
| **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 trong C++

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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:

[**Ví dụ**](https://vietjack.com/cplusplus/toan_tu_so_sanh_bit_trong_cplusplus.jsp)

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

## 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.

Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

[**Ví dụ**](https://vietjack.com/cplusplus/thu_tu_uu_tien_toan_tu_trong_cplusplus.jsp)

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

### Vòng lặp

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

C++ hỗ trợ vòng lặp sau đây. Click chuột vào link để xem chi tiết.

|  |  |
| --- | --- |
| **Vòng lặp** | **Miêu tả** |
| [**Vòng lặp while trong C++**](https://vietjack.com/cplusplus/vong_lap_while_trong_cplusplus.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 trong C++**](https://vietjack.com/cplusplus/vong_lap_for_trong_cplusplus.jsp) | Thực thi một dãy các lệnh nhiều lần và tóm tắt đoạn code mà quản lý biến vòng lặp. |
| [**Vòng lặp do...while trong C++**](https://vietjack.com/cplusplus/vong_lap_do_while_trong_cplusplus.jsp) | Giống lệnh While, ngoại trừ ở điểm là nó kiểm tra điều kiện ở cuối thân vòng lặp. |
| [**Lồng vòng lặp trong C++**](https://vietjack.com/cplusplus/long_vong_lap_trong_cplusplus.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 trong C++

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.

|  |  |
| --- | --- |
| **Lệnh điều khiển** | **Miêu tả** |
| [**Lệnh break trong C++**](https://vietjack.com/cplusplus/lenh_break_trong_cplusplus.jsp) | Kết thúc **vòng lặp** hoặc lệnh **switch** và chuyển sang thực thi vòng lặp hoặc lệnh switch ngay sau nó. |
| [**Lệnh continue trong C++**](https://vietjack.com/cplusplus/lenh_continue_trong_cplusplus.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 trong C++**](https://vietjack.com/cplusplus/lenh_goto_trong_cplusplus.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 trong C++

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 <iostream>

using namespace std;

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. Bạn có thể có một biểu thức khởi tạo và biểu thức tăng, giảm, nhưng các lập trình viên C++ thường sử dụng for(;;) để biểu thị một vòng lặp vô hạn.

### Lệnh IF, Else, Switch

Các cấu trúc điều khiển luồng yêu cầu lập trình viên xác định một hoặc nhiều điều kiện để được 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 liệt kê các lệnh điều khiển luồng được cung cấp bởi C++. Bạn click vào link để thấy chi tiết.

|  |  |
| --- | --- |
| **Lệnh** | **Miêu tả** |
| [**Lệnh if trong C++**](https://vietjack.com/cplusplus/lenh_if_trong_cplusplus.jsp) | Một lệnh if bao gồm một biểu thức logic theo sau bởi một hoặc nhiều lệnh khác. |
| [**Lệnh if...else trong C++**](https://vietjack.com/cplusplus/lenh_if_else_trong_cplusplus.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. |
| [**Lệnh switch trong C++**](https://vietjack.com/cplusplus/lenh_switch_trong_cplusplus.jsp) | Một lệnh switch cho phép kiểm tra điều kiện của một biến trước khi thực thi các lệnh. |
| [**Lồng các lệnh if trong C++**](https://vietjack.com/cplusplus/long_lenh_if_trong_cplusplus.jsp) | Bạn có thể sử dụng lệnh if hoặc else if bên trong lệnh if hoặc else if khác. |
| [**Lồng các lệnh switch trong C++**](https://vietjack.com/cplusplus/long_lenh_switch_trong_cplusplus.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 ? :**](https://vietjack.com/cplusplus/toan_tu_dieu_kien_trong_cplusplus.jsp) trong chương trước mà có thể được dùng để thay thế cho lệnh **if...else**. Nó có mẫu chung như sau:

Exp1 ? Exp2 : Exp3;

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.

Một hàm là một nhóm các lệnh đi cùng nhau để thực hiện một nhiệm vụ. Mỗi chương trình C/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/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/C++

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

Kieu\_tra\_ve Ten\_ham( Danh sach tham so )

{

Than ham

}

Một định nghĩa hàm trong ngôn ngữ C/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.
* **Danh sách 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.

Quảng cáo

Ví dụ: Sau đây phần code cho một hàm có tên gọi là **max()**. Hàm này có 2 tham số: so1 và so2 và trả về giá trị lớn nhất giữa hàm số:

// ham tra ve so lon nhat cua hai so

int max(int so1, int so2)

{

// Khai bao bien cuc bo

int result;

if (so1 > so2)

result = so1;

else

result = so2;

return result;

}

Khai báo hàm trong C/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 max(), dưới đây là câu khai báo hàm:

int max(int so1, int so2);

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

int max(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/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 <iostream>

using namespace std;

// khai bao ham

int max(int so1, int so2);

int main ()

{

// Khai bao bien cuc bo:

int a = 100;

int b = 200;

int ketqua;

// goi ham de tim gia tri lon nhat.

ketqua = max(a, b);

cout << "Gia tri lon nhat la: " << ketqua << endl;

return 0;

}

// ham tra ve so lon nhat cua hai so

int max(int so1, int so2)

{

// Khai bao bien cuc bo

int result;

if (so1 > so2)

result = so1;

else

result = so2;

return result;

}

Mình giữ giá trị hàm max() trong hàm main vào biến ketqua. Khi chạy chương trình C/C++ trên sẽ cho kết quả sau:

Gia tri lon nhat la: 200

## Tham số của hàm trong C/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 hàm bởi giá trị trong C/C++**](https://vietjack.com/cplusplus/goi_ham_boi_gia_tri_trong_cplusplus.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 hàm bởi con trỏ trong C/C++**](https://vietjack.com/cplusplus/goi_ham_boi_con_tro_trong_cplusplus.jsp) | Phương thức này sao chép địa chỉ của tham số vào trong biến chính thức. Bên trong hàm này, địa chỉ này được sử dụng để truy cập tham số thực sự được sử dụng trong lời gọi hàm. |
| [**Gọi hàm bởi tham chiếu trong C/C++**](https://vietjack.com/cplusplus/goi_ham_boi_tham_chieu_trong_cplusplus.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/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 max() là dùng phương thức tương tự.

**Giá trị mặc định cho các tham số trong C/C++**

Khi bạn định nghĩa một hàm, bạn có thể xác định một giá trị mặc định cho mỗi tham số cuối cùng. Giá trị này sẽ được sử dụng nếu tham số tương ứng là để trống bên trái khi gọi hàm đó.

Việc này được thực hiện bởi sử dụng toán tử gán và gán các giá trị cho các tham số trong định nghĩa hàm. Nếu một giá trị cho tham số đó không được truyền khi hàm được gọi, thì giá trị mặc định đã cung cấp sẽ được sử dụng, nhưng nếu một giá trị đã được xác định, thì giá trị mặc định này bị bỏ qua và, thay vào đó, giá trị đã truyền được sử dụng. Bạn theo dõi ví dụ sau:

#include <iostream>

using namespace std;

int sum(int a, int b=20)

{

int ketqua;

ketqua = a + b;

return (ketqua);

}

int main ()

{

// Khai bao bien cuc bo:

int a = 100;

int b = 200;

int ketqua;

// goi ham de tinh tong hai so.

ketqua = sum(a, b);

cout << "Tong gia tri la: " << ketqua << endl;

// goi ham mot lan nua.

ketqua = sum(a);

cout << "Tong gia tri la: " << ketqua << endl;

return 0;

}

Chạy chương trình C/C++ trên sẽ cho kết quả sau:

Tong gia tri la: 300

Tong gia tri la: 120

Theo dõi chúng tôi miễn phí trên mạng xã hội facebook và youtube:

### Number

Thông thường, khi chúng ta làm việc với Number (các kiểu giá trị số), chúng ta sử dụng các kiểu dữ liệu gốc như int, short, long, float và double, …. Các kiểu dữ liệu số, về giá trị có thể và dãy giá trị của chúng, đã được bàn luận trong chương Kiểu dữ liệu trong C++.

Định nghĩa Number trong C++

#include <iostream>

using namespace std;

int main ()

{

// phan dinh nghia cac so:

short s;

int i;

long l;

float f;

double d;

// phep gan cho cac so;

s = 10;

i = 1000;

l = 1000000;

f = 230.47;

d = 30949.374;

// in cac so;

cout << "short s la: " << s << endl;

cout << "int i la: " << i << endl;

cout << "long l la: " << l << endl;

cout << "float f la: " << f << endl;

cout << "double d la: " << d << endl;

return 0;

}

Quảng cáo

Khi code trên được biên dịch và thực thi, nó cho kết quả sau:

short s la: 10

int i la: 1000

long l la: 1000000

float f la: 230.47

double d la: 30949.4

**Hàm toán học trong C++**

Bên cạnh các hàm đa dạng bạn có thể tạo, C++ cũng bao gồm một số hàm toán học hữu ích cho bạn sử dụng. Những hàm này có sẵn trong các thư viện C và C++ chuẩn, và được gọi là các hàm **built-in**. Đây là các hàm mà có thể được bao trong chương trình của bạn và sau đó sử dụng.

Để sử dụng các hàm này, bạn cần bao header file là **<cmath>**.

|  |  |
| --- | --- |
| **STT** | **Hàm & Mục đích** |
| 1 | **double cos(double);**  Hàm này trả về cosin của một góc (dạng một double) |
| 2 | **double sin(double);**  Hàm này trả về sin của một góc (dạng một double) |
| 3 | **double tan(double);**  Hàm này trả về tang của một góc (dạng một double) |
| 4 | **double log(double);**  Hàm này trả về logarit tự nhiên (ln) của số đó |
| 5 | **double pow(double, double);**  Hàm mũ với cơ số là số double đầu tiên và số mũ là double thứ hai |
| 6 | **double hypot(double, double);**  Nếu bạn truyền độ dài của hai cạnh của tam giác (lần lượt là các số double), nó sẽ trả về độ dài cạnh huyền |
| 7 | **double sqrt(double);**  Trả về căn bậc hai của số double |
| 8 | **int abs(int);**  Trả về trị tuyệt đối của int |
| 9 | **double fabs(double);**  Trả về trị tuyệt đối của bất kỳ số double nào |
| 10 | **double floor(double);**  Tìm số integer mà nhỏ hơn hoặc bằng tham số đã truyền cho nó |

Ví dụ sau minh họa một số hàm toán học trong C++:

#include <iostream>

#include <cmath>

using namespace std;

int main ()

{

// phan dinh nghia cac so:

short s = 10;

int i = -1000;

long l = 100000;

float f = 230.47;

double d = 200.374;

// cac hoat dong toan hoc;

cout << "sin(d) co gia tri la: " << sin(d) << endl;

cout << "abs(i) co gia tri la: " << abs(i) << endl;

cout << "floor(d) co gia tri la: " << floor(d) << endl;

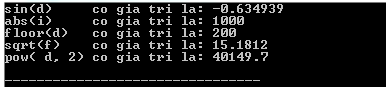
cout << "sqrt(f) co gia tri la: " << sqrt(f) << endl;

cout << "pow( d, 2) co gia tri la: " << pow(d, 2) << endl;

return 0;

}

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:



## Số ngẫu nhiên (Random Number) trong C++

Trong một số trường hợp, bạn muốn tạo một số ngẫu nhiên (random number). Có hai hàm có thể giúp bạn thực hiện việc này. Hàm đầu tiên là **rand()** được định nghĩa trong thư viện <cstdlib>: sẽ chỉ trả về một số ngẫu nhiên giả, mà có thể không thay đổi qua các lần chạy chương trình. Để giải quyết điểm bất thường này, chúng ta sẽ sử dụng hàm **srand()** trong thư viện <ctime>.

Ví dụ sau sẽ tạo vài số ngẫu nhiên trong C++. Ví dụ có sử dụng hàm time() để lấy số giây trên System time của bạn. Giá trị trả về từ time là qua scrand, bạn lưu ý là số ngẫu nhiên được tạo ra trước lời gọi rand.

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

int main ()

{

int i,j;

srand( (unsigned)time( NULL ) );

/\* tao 10 so ngau nhien. \*/

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

{

j= rand();

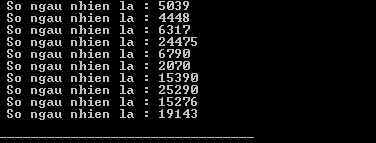
cout <<" So ngau nhien la : " << j << endl;

}

return 0;

}

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:



### Mảng

Ngôn ngữ lập trình C/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 so0, so1,… và so99, bạn có thể khai báo một mảng các giá trị như so[0], so[1] và … so[99] để biểu diễn các giá trị riêng biệt. Một thành viên cụ thể của mảng có thể được truy cập qua index (chỉ số).

**Khai báo mảng trong C/C++**

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/C++. Ví dụ, khai báo một mảng 10 phần tử gọi là balance với kiểu double, sử dụng câu lệnh sau đây:

char sinhvien[10];

Quảng cáo

Khởi tạo mảng trong C/C++

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

int hanghoa[5] = {45, 34, 29, 67, 49};

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

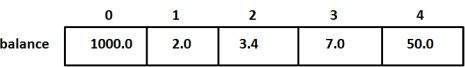
Nếu bạn bỏ sót kích cỡ mảng thì mảng đó đủ lớn để giữ các giá trị được khởi tạo: Bạn sẽ tạo chính xác một chuỗi có giá trị giống hệt chuỗi bên trên bằng cách gán từng phần tử một. Dưới đây là một ví dụ khi gán giá trị cho một phần tử của mảng:

int hanghoa[] = {45, 34, 29, 67, 49};

Bạn có thể tạo ra cùng một mảng giống như đã làm trong ví dụ trước.

hanghoa[4] = 50;

Câu lệnh bên trên gán giá trị thứ 5 của mảng giá trị 50.0. 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/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 hocphi = hocphik60[55];

Câu lệnh trên lấy phần tử thứ 56 của mảng và gán giá trị này cho biến hocphi. Dưới đây là một ví dụ về việc sử dụng với tất cả mô tả bên trên:

#include <iostream>

using namespace std;

#include <iomanip>

using std::setw;

int main ()

{

int n[ 10 ]; // n la mot mang gom 10 so nguyen

// khoi tao gia tri cac phan tu cua mang n la 0

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

{

n[ i ] = i + 100; // thiet lap phan tu tai vi tri i la i + 100

}

cout << "Phan tu thu:" << setw( 13 ) << "Gia tri la:" << endl;

// hien thi gia tri cua moi phan tu

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

{

cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;

}

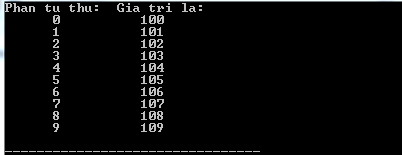
return 0;

}

Quảng cáo

Chương trình này sử dụng hàm **setw(so\_nguyen)** trong C/C++ để định dạng output. Tại đây, tham số **so\_nguyen** là một số chỉ độ rộng của kết quả mà bạn muốn hiển thị. Chẳng hạn, với so\_nguyen là 3 tức là bạn dành 3 vị trí để in kết quả, nếu kết quả cần hiển thị là thừa thì nó sẽ bị cắt bớt, nếu thiếu thì chèn thêm khoảng trống vào. Hàm setw() được dùng cho cả **cout và cin**.

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:



## Chi tiết về mảng trong C/C++

Mảng là một phần rất quan trọng trong ngôn ngữ C/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/C++:

Mảng là một phần rất quan trọng trong ngôn ngữ C/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/C++:

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Mảng đa chiều trong C/C++**](https://vietjack.com/cplusplus/mang_da_chieu_trong_cplusplus.jsp) | C/C++ hỗ trợ các mảng đa chiều. Mẫu đơn giản nhất của mảng này là mảng hai chiều |
| [**Con trỏ tới một mảng trong C/C++**](https://vietjack.com/cplusplus/con_tro_toi_mang_trong_cplusplus.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 |
| [**Truyền mảng tới hàm như là tham số trong C/C++**](https://vietjack.com/cplusplus/truyen_mang_toi_ham_trong_cplusplus.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 hàm trong C/C++**](https://vietjack.com/cplusplus/tra_ve_mang_tu_ham_trong_cplusplus.jsp) |  |

### Chuỗi(String)

Dạng chuỗi này bắt nguồn từ ngôn ngữ C và tiếp tục được hỗ trợ trong C/C++. 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'.

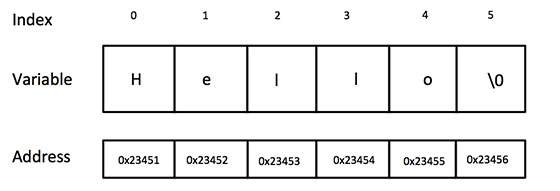
Phần khai báo và khởi tạo dưới đây tạo ra một chuỗi bao gồm một từ "Hello". Để giữ các giá trị null tại cuối của mảng, cỡ của mảng các ký tự bao gồm một chuỗi phải nhiều hơn số lượng các ký tự trong từ khóa "Hello".

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 <iostream>

using namespace std;

int main ()

{

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

cout << "Khi gap nhau, chung ta noi: ";

cout << loiChao << endl;

return 0;

}

Quảng cáo

Khi đoạn code trên được biên dịch và thực hiện, kết quả in ra sẽ như sau:

Khi gap nhau, chung ta noi: Hello

Ngôn ngữ C/C++ hỗ trợ nhiều hàm đa dạng 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 <iostream>

#include <cstring>

using namespace std;

int main ()

{

char chuoi1[10] = "Hello";

char chuoi2[10] = "Christmas";

char chuoi3[10];

int len ;

// sao chep chuoi1 vao trong chuoi3

strcpy( chuoi3, chuoi1);

cout << "strcpy( chuoi3, chuoi1) : " << chuoi3 << endl;

// noi hai chuoi: chuoi1 va chuoi2

strcat( chuoi1, chuoi2);

cout << "strcat( chuoi1, chuoi2): " << chuoi1 << endl;

// tong do dai cua chuoi1 mot sau khi thuc hien noi chuoi

len = strlen(chuoi1);

cout << "Dung ham strlen(chuoi1) de tinh do dai chuoi1: " << len << endl;

return 0;

}

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

Chuỗi trong C/C++

**Quảng cáo**

**Lớp String trong C/C++**

Thư viện chuẩn C/C++ cung cấp một kiểu lớp **String** mà hỗ trợ tất cả hoạt động liên quan tới chuỗi đã đề cập ở trên, và bổ sung thêm nhiều tính năng nữa. Chúng ta sẽ học lớp này trong Thư viện chuẩn C/C++ (C++ Standard Library), nhưng lúc này, chúng ta xem xét ví dụ sau:

Lúc này, có thể bạn không hiểu ví dụ này, bởi vì chúng ta chưa bàn luận về **Lớp và Đối tượng** trong C/C++. Vì thế, bạn quan sát và ghi nhớ chúng tới khi bạn đã hiểu các khái niệm về Hướng đối tượng được trình bày ở chương sau đó.

#include <iostream>

#include <string>

using namespace std;

int main ()

{

string chuoi1 = "Hello";

string chuoi2 = "Christmas";

string chuoi3;

int len ;

// sao chep chuoi1 vao trong chuoi3

chuoi3 = chuoi1;

cout << "Bay gio chuoi3 la: " << chuoi3 << endl;

// noi hai chuoi: chuoi1 va chuoi2

chuoi3 = chuoi1 + chuoi2;

cout << "chuoi1 + chuoi2 co ket qua la: " << chuoi3 << endl;

// tong do dai cua chuoi3 mot sau khi thuc hien noi chuoi

len = chuoi3.size();

cout << "Tinh do dai voi ham chuoi3.size() : " << len << endl;

return 0;

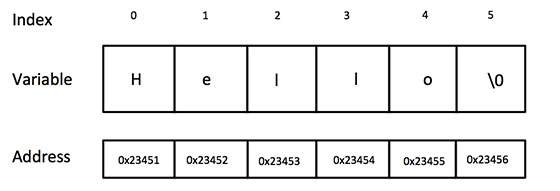
}

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

Lớp String trong C/C++

Theo dõi chúng tôi miễn phí trên mạng xã hội facebook và youtube:

Các bạn có thể mua thêm khóa học JAVA CORE ONLINE VÀ ỨNG DỤNG cực hay, giúp các bạn vượt qua các dự án trên trường và đi thực tập Java. Khóa học có giá chỉ 300K, nhằm ưu đãi, tạo điều kiện cho sinh viên cho thể mua khóa học.



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 <iostream>

using namespace std;

int main ()

{

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

cout << "Khi gap nhau, chung ta noi: ";

cout << loiChao << endl;

return 0;

}

Quảng cáo

Khi đoạn code trên được biên dịch và thực hiện, kết quả in ra sẽ như sau:

Khi gap nhau, chung ta noi: Hello

Ngôn ngữ C/C++ hỗ trợ nhiều hàm đa dạng 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 <iostream>

#include <cstring>

using namespace std;

int main ()

{

char chuoi1[10] = "Hello";

char chuoi2[10] = "Christmas";

char chuoi3[10];

int len ;

// sao chep chuoi1 vao trong chuoi3

strcpy( chuoi3, chuoi1);

cout << "strcpy( chuoi3, chuoi1) : " << chuoi3 << endl;

// noi hai chuoi: chuoi1 va chuoi2

strcat( chuoi1, chuoi2);

cout << "strcat( chuoi1, chuoi2): " << chuoi1 << endl;

// tong do dai cua chuoi1 mot sau khi thuc hien noi chuoi

len = strlen(chuoi1);

cout << "Dung ham strlen(chuoi1) de tinh do dai chuoi1: " << len << endl;

return 0;

}

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

Chuỗi trong C/C++

**Quảng cáo**

**Lớp String trong C/C++**

Thư viện chuẩn C/C++ cung cấp một kiểu lớp **String** mà hỗ trợ tất cả hoạt động liên quan tới chuỗi đã đề cập ở trên, và bổ sung thêm nhiều tính năng nữa. Chúng ta sẽ học lớp này trong Thư viện chuẩn C/C++ (C++ Standard Library), nhưng lúc này, chúng ta xem xét ví dụ sau:

Lúc này, có thể bạn không hiểu ví dụ này, bởi vì chúng ta chưa bàn luận về **Lớp và Đối tượng** trong C/C++. Vì thế, bạn quan sát và ghi nhớ chúng tới khi bạn đã hiểu các khái niệm về Hướng đối tượng được trình bày ở chương sau đó.

#include <iostream>

#include <string>

using namespace std;

int main ()

{

string chuoi1 = "Hello";

string chuoi2 = "Christmas";

string chuoi3;

int len ;

// sao chep chuoi1 vao trong chuoi3

chuoi3 = chuoi1;

cout << "Bay gio chuoi3 la: " << chuoi3 << endl;

// noi hai chuoi: chuoi1 va chuoi2

chuoi3 = chuoi1 + chuoi2;

cout << "chuoi1 + chuoi2 co ket qua la: " << chuoi3 << endl;

// tong do dai cua chuoi3 mot sau khi thuc hien noi chuoi

len = chuoi3.size();

cout << "Tinh do dai voi ham chuoi3.size() : " << len << endl;

return 0;

}

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:

Lớp String trong C/C++

### Con trỏ

**[Con trỏ trong C /C++]** **Con trỏ - Pointer** trong ngôn ngữ C /C++ rất dễ học và thú vị. Một vài tác vụ trong ngôn ngữ C /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 /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 <iostream>

using namespace std;

int main ()

{

int bien1;

char bien2[10];

cout << "Dia chi cua bien1 la: ";

cout << &bien1 << endl;

cout << "Dia chi cua bien2 la: ";

cout << &bien2 << endl;

return 0;

}

Chạy chương trình C /C++ trên sẽ cho kết quả như hình sau:

Con trỏ trong C /C++

**Con trỏ là gì?**

Một **con trỏ** 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 **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 \*sv; // tro toi mot gia tri nguyen

double \*nv; // tro toi mot gia tri double

float \*luong; // tro toi mot gia tri float

char \*ten // tro toi mot ky tu

Kiểu dữ liệu thực sự của giá trị của tất cả các con trỏ, có thể là integer, float, character, hoặc kiểu khác, là giống như, một số 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.

**Quảng cáo**

**Cách sử dụng con trỏ trong C /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 <iostream>

using namespace std;

int main ()

{

int bien1 = 15000; // khai bao bien.

int \*sv; // bien con tro sv

sv = &bien1; // luu tru dia chi cua bien1 vao bien con tro sv

cout << "Gia tri cua bien1 la: ";

cout << bien1 << endl;

// In dia chi duoc luu tru trong bien con tro sv

cout << "Dia chi duoc luu tru trong bien con tro sv la: ";

cout << sv << endl;

// Truy cap gia tri co san tai dia chi cua bien con tro

cout << "Gia tri cua \*sv la: ";

cout << \*sv << endl;

return 0;

}

Chạy chương trình C /C++ trên sẽ cho kết quả như hình sau:

Sử dụng Con trỏ trong C /C++

**Chi tiết về con trỏ trong C /C++**

Con trỏ có nhiều khái niệm nhưng cũng khá dễ để học và rất quan trọng trong lập trình ngôn ngữ C /C++. Dưới đây là những khái niệm quan trọng về con trỏ trong C /C++, được trình bày rõ ràng cho bạn, khi bạn click vào link tương ứng.

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Con trỏ Null trong C /C++**](https://vietjack.com/cplusplus/con_tro_null_trong_cplusplus.jsp) | C /C++ hỗ trợ con trỏ null, là một constant với giá trị là 0 được định nghĩa trong một số thư viện chuẩn |
| [**Con trỏ số học trong C /C++**](https://vietjack.com/cplusplus/con_tro_so_hoc_trong_cplusplus.jsp) | Có 4 toán tử đại số mà có thể được sử dụng trên các con trỏ là: ++, --, +, - |
| [**Con trỏ và mảng trong C /C++**](https://vietjack.com/cplusplus/con_tro_va_mang_trong_cplusplus.jsp) | Có mối liên hệ gì giữa con trỏ và mảng. Bạn kiểm tra ngay. |
| [**Mảng con trỏ trong C /C++**](https://vietjack.com/cplusplus/mang_con_tro_trong_cplusplus.jsp) | Bạn có thể định nghĩa các mảng để giữ các con trỏ. |
| [**Con trỏ tới con trỏ trong C /C++**](https://vietjack.com/cplusplus/con_tro_tro_toi_con_tro_trong_cplusplus.jsp) | C /C++ cho phép bạn trỏ tới một con trỏ ... |
| [**Truyền con trỏ tới hàm trong C /C++**](https://vietjack.com/cplusplus/truyen_con_tro_toi_ham_trong_cplusplus.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 /C++**](https://vietjack.com/cplusplus/tra_ve_con_tro_tu_ham_trong_cplusplus.jsp) | C /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. |

### Tham chiếu

Một biến tham chiếu là một **alias**, đó là tên khác cho một biến đã đang tồn tại. Khi một tham chiếu được khởi tạo với một biến, thì: hoặc tên biến hoặc tên tham chiếu có thể được sử dụng để tham chiếu tới biến đó.

## Tham chiếu và con trỏ trong C++

Sử dụng tham chiếu có thể thường gây bối rối cho bạn như với con trỏ, tuy nhiên, có 3 điểm khác nhau giữa tham chiếu và con trỏ trong C++:

* Bạn không thể có tham chiếu NULL. Bạn luôn luôn phải giả sử rằng một tham chiếu được kết nối với một phần của kho lưu trữ.
* Khi một tham chiếu được khởi tạo cho một đối tượng, nó không thể bị thay đổi để tham chiếu tới đối tượng khác. Các con trỏ có thể được trỏ tới đối tượng khác tại bất kỳ thời điểm nào.
* Một tham chiếu phải được khởi tạo khi nó được tạo. Các con trỏ có thể được tạo tại bất kỳ thời điểm nào.

## Tạo tham chiếu trong C++

Bạn coi một tên biến như là một label (một nhãn) được đính kèm với vị trí biến trong bộ nhớ. Sau đó, bạn có thể nghĩ về tham chiếu như là label thứ hai được đính kèm với vị trí bộ nhớ đó. Vì thế, bạn có thể truy cập nội dung của biến thông qua: hoặc tên biến ban đầu hoặc tham chiếu. Giả sử chúng ta có ví dụ sau:

int i = 19;

Bạn có thể khai báo các biến tham chiếu cho i như sau:

int& r = i;

Đọc & trong các khai báo này là **Tham chiếu**. Vì thế, trong khai báo đầu tiên, r là tham chiếu integer được khởi tạo cho i và trong tham chiếu thứ hai, s là tham chiếu double được khởi tạo cho d. Ví dụ sau sử dụng các tham chiếu trong C++:

#include <iostream>

using namespace std;

int main ()

{

// khai bao cac bien

int i;

double d;

// khai bao cac bien tham chieu

int& r = i;

double& s = d;

i = 5;

cout << "Gia tri cua i la: " << i << endl;

cout << "Gia tri cua tham chieu toi i la: " << r << endl;

d = 11.7;

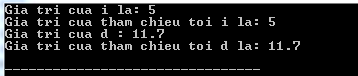
cout << "Gia tri cua d : " << d << endl;

cout << "Gia tri cua tham chieu toi d la: " << s << endl;

return 0;

}

Chạy chương trình C++ trên sẽ cho kết quả như hình sau:



Tham chiếu trong C++ thường được sử dụng cho danh sách các tham số hàm và hàm trả về các giá trị. Sau đây là hai khái niệm quan trọng liên quan tới tham chiếu trong C++, bạn sẽ thấy chi tiết khi click vào link:

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Truyền tham số bởi tham chiếu trong C++**](https://vietjack.com/cplusplus/truyen_tham_so_boi_tham_chieu_trong_cplusplus.jsp) | C++ hỗ trợ truyền các tham chiếu như là các tham số hàm mà an toàn hơn so với các tham số |
| [**Trả về giá trị bởi tham chiếu trong C++**](https://vietjack.com/cplusplus/tra_ve_gia_tri_boi_tham_chieu_trong_cplusplus.jsp) | Bạn có thể trả về tham chiếu từ một hàm C++ giống như bất kỳ kiểu dữ liệu khác có thể được trả về |

### Struct

**Struct trong C/C++** Các mảng trong C/C++ cho phép bạn định nghĩa một vài loại biến có thể giữ giá trị của một vài thành viên cùng kiểu dữ liêu. Nhưng **structure - cấu trúc** là một loại dữ liệu khác trong ngôn ngữ lập trình C/C++, cho phép bạn kết hợp các dữ liệu khác kiểu nhau.

Cấu trúc được sử dụng để biểu diễn một bản ghi. Giả sử bạn muốn lưu trữ giá trị của một quyển sách trong thư viện của bạn. Bạn có thể lưu trữ các thuộc tính của sách sau đây:

* Tiêu đề
* Tác giả
* Chủ đề
* Book ID

Định nghĩa một cấu trúc trong C++

Để định nghĩa cấu trúc, bạn phải sử dụng câu lệnh **struct**.

struct [ten cau truc]

{

phan dinh nghia thanh vien;

phan dinh nghia thanh vien;

...

phan dinh nghia thanh vien;

} [mot hoac nhieu bien cau truc];

Quảng cáo

Ở đây, **ten cau truc** có thể tùy ý và một thành viên định nghĩa là các biến thường như int i, float j hoặc một định nghĩa biến khác …. Tại phần cuối cùng của định nghĩa cấu trúc, trước dấu chấm phẩy, bạn có thể xác định một hoặc nhiều biến cấu trúc (tùy chọn). Dưới đây là cách khai báo biến cấu trúc Book:

struct Books

{

char tieude[50];

char tacgia[50];

char chude[100];

int book\_id;

}book;

**Truy cập các thành viên của cấu trúc trong C++**

Để truy cập bất kỳ thành viên nào của cấu trúc, bạn sử dụng **toán tử truy cập phần tử (.)**. Toán tử truy cập thành viên cấu trúc được mã hóa là dấu chấm giữa tên biến cấu trúc và thành viên cấu trúc mà bạn muốn truy cập. Bạn sẽ sử dụng từ khóa **struct** để định nghĩa các biến của kiểu cấu trúc. Dưới đây là ví dụ cho cách sử dụng cấu trúc trong C++:

#include <iostream>

#include <cstring>

using namespace std;

struct Books

{

char tieude[50];

char tacgia[50];

char chude[100];

int book\_id;

};

int main( )

{

struct Books QuyenSach1; // Declare QuyenSach1 of type Book

struct Books QuyenSach2; // Declare QuyenSach2 of type Book

// chi tiet ve quyen sach thu nhat

strcpy( QuyenSach1.tieude, "Ngon ngu Lap trinh C++");

strcpy( QuyenSach1.tacgia, "Pham Van At");

strcpy( QuyenSach1.chude, "Lap trinh");

QuyenSach1.book\_id = 1225;

// chi tiet ve quyen sach thu hai

strcpy( QuyenSach2.tieude, "Toi thay hoa vang tren co xanh");

strcpy( QuyenSach2.tacgia, "Nguyen Nhat Anh");

strcpy( QuyenSach2.chude, "Van hoc");

QuyenSach2.book\_id = 3214;

// in thong tin ve QuyenSach1

cout << "Tieu de cua Quyen sach thu nhat la: " << QuyenSach1.tieude <<endl;

cout << "Tac gia cua Quyen sach thu nhat la: " << QuyenSach1.tacgia <<endl;

cout << "Chu de cua Quyen sach thu nhat la: " << QuyenSach1.chude <<endl;

cout << "ID cua Quyen sach thu nhat la: " << QuyenSach1.book\_id <<endl;

cout << "\n\n===================================================================\n\n" <<endl;

// in thong tin ve QuyenSach2

cout << "Tieu de cua Quyen sach thu hai la: " << QuyenSach2.tieude <<endl;

cout << "Tac gia cua Quyen sach thu hai la: " << QuyenSach2.tacgia <<endl;

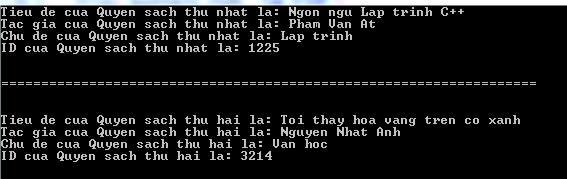
cout << "Chu de cua Quyen sach thu hai la: " << QuyenSach2.chude <<endl;

cout << "ID cua Quyen sach thu hai la: " << QuyenSach2.book\_id <<endl;

return 0;

}

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



**Cấu trúc dưới dạng tham số hàm trong C++**

Bạn có thể truyền một cấu trúc như một tham số của hàm theo cách khá giống như khi bạn truyền bất kỳ biến hay con trỏ khác. Bạn sẽ truy cập biến cấu trúc theo cách tương tự như bạn đã truy cập trong ví dụ trên:

#include <iostream>

#include <cstring>

using namespace std;

void inthongtin( struct Books book );

struct Books

{

char tieude[50];

char tacgia[50];

char chude[100];

int book\_id;

};

int main( )

{

struct Books QuyenSach1; // Khai bao QuyenSach1 la cua kieu Books

struct Books QuyenSach2; // Khai bao QuyenSach2 la cua kieu Books

// chi tiet ve quyen sach thu nhat

strcpy( QuyenSach1.tieude, "Ngon ngu Lap trinh C++");

strcpy( QuyenSach1.tacgia, "Pham Van At");

strcpy( QuyenSach1.chude, "Lap trinh");

QuyenSach1.book\_id = 1225;

// chi tiet ve quyen sach thu hai

strcpy( QuyenSach2.tieude, "Toi thay hoa vang tren co xanh");

strcpy( QuyenSach2.tacgia, "Nguyen Nhat Anh");

strcpy( QuyenSach2.chude, "Van hoc");

QuyenSach2.book\_id = 3214;

// in thong tin ve QuyenSach1

inthongtin( QuyenSach1 );

// in thong tin ve QuyenSach2

inthongtin( QuyenSach2 );

return 0;

}

void inthongtin( struct Books book )

{

cout << "Tieu de sach: " << book.tieude <<endl;

cout << "Tac gia: " << book.tacgia <<endl;

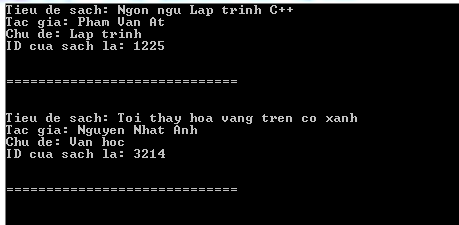
cout << "Chu de: " << book.chude <<endl;

cout << "ID cua sach la: " << book.book\_id <<endl;

cout << "\n\n========================================\n\n" <<endl;

}

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



**Con trỏ tới cấu trúc trong C++**

Bạn có thể định nghĩa con trỏ cấu trúc theo cách tương tự bạn định nghĩa con trỏ tới bất kỳ biến nào khác như sau:

struct Books \*contro\_struct;

Bây giờ bạn có thể lưu địa chỉ của biến cấu trúc trong biến con trỏ được định nghĩa ở trên. Để tìm địa chỉ của một biến cấu trúc, đặt toán tử & trước tên cấu trúc như sau:

contro\_struct = &QuyenSach1;

Để truy cập vào thành viên của một structure sử dụng con trỏ tới structure đó, bạn phải sử dụng toán tử -> như sau:

contro\_struct->tieude;

Bây giờ chúng ta viết lại ví dụ trên sử dụng con trỏ cấu trúc, hy vọng điều này sẽ dễ dàng cho bạn để hiểu khái niệm này:

#include <iostream>

#include <cstring>

using namespace std;

void inthongtin( struct Books \*book );

struct Books

{

char tieude[50];

char tacgia[50];

char chude[100];

int book\_id;

};

int main( )

{

struct Books QuyenSach1; // Khai bao QuyenSach1 la cua kieu Books

struct Books QuyenSach2; // Khai bao QuyenSach2 la cua kieu Book

// thong tin chi thiet ve quyen sach thu nhat

strcpy( QuyenSach1.tieude, "Ngon ngu Lap trinh C++");

strcpy( QuyenSach1.tacgia, "Pham Van At");

strcpy( QuyenSach1.chude, "Lap trinh");

QuyenSach1.book\_id = 1225;

// thong tin chi thiet ve quyen sach thu hai

strcpy( QuyenSach2.tieude, "Toi thay hoa vang tren co xanh");

strcpy( QuyenSach2.tacgia, "Nguyen Nhat Anh");

strcpy( QuyenSach2.chude, "Van hoc");

QuyenSach2.book\_id = 3214;

// in thong tin cua QuyenSach1, bang cach truyen dia chi cua cau truc

inthongtin( &QuyenSach1 );

// in thong tin cua QuyenSach2, bang cach truyen dia chi cua cau truc

inthongtin( &QuyenSach2 );

return 0;

}

// Ham nay chap nhan con tro toi cau truc lam tham so.

void inthongtin( struct Books \*book )

{

cout << "Tieu de sach: " << book->tieude <<endl;

cout << "Tac gia: " << book->tacgia <<endl;

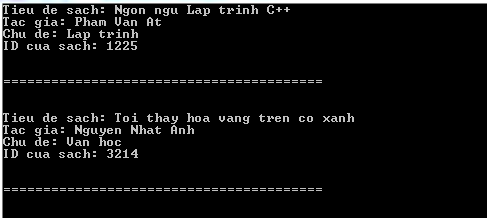
cout << "Chu de: " << book->chude <<endl;

cout << "ID cua sach: " << book->book\_id <<endl;

cout << "\n\n========================================\n\n" <<endl;

}

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



## Từ khóa typedef trong C++

Có một cách dễ dàng hơn để định nghĩa các cấu trúc hoặc bạn có thể "alias" các kiểu bạn tạo. Ví dụ:

typedef struct

{

char tieude[50];

char tacgia[50];

char chude[100];

int book\_id;

}Books;

Lúc này, bạn có thể sử dụng **Books** một cách trực tiếp để định nghĩa các biến của kiểu cấu trúc **Books** mà không sử dụng từ khóa struct. Sau đây là ví dụ:

Books QuyenSach1, QuyenSach2;

Bạn có thể sử dụng từ khóa **typedef** trong C++ cho các dạng không phải cấu trúc, như sau:

typedef long int \*pint32;

pint32 x, y, z;

Với x, y và z là tất cả con trỏ tới long int.