C tutorial

Lập trình C là một mục đích chung, thủ tục, câu lệnh ngôn ngữ lập trình máy tính được phát triển vào năm 1972 bởi Dennis M. Ritchie tại phòng thí nghiệm Bell Tellephone để phát triển hệ điều hành UNIX. C là ngôn ngữ máy tính được sử dụng rộng rãi nhất. Nó giữ vị trí số một tại thang đo mức độ nổi tiếng cùng với ngôn ngữ lập trình Java, với Java cũng nổi tiếng ngang bằng và được sử dụng rộng rãi nhất trong lập trình phần mềm hiện đại.

**Tại sao học lập trình C ?**

Ngôn ngữ lập trình C là bắt buộc cho học sinh và chuyên gia làm việc để trở thành một Kỹ sư phần mềm đặc biệt khi làm việc cho mảng phát triển phần mềm.

\*Lợi ích khi học lập trình C:

-Dễ dàng học.

-Ngôn ngữ có cấu trúc.

-Cung cấp chương trình hiệu quả.

-Có thể xử lý những hoạt động cơ bản.

-Có thể biên dịch đa dạng nền tảng máy tính.

**Sự thật về C**

-Cđượcphát minh để viết một hệ điều hành gọi là UNIX.

-C kế thừa từ Ngôn ngữ B được giới thiệu vào khoảng những năm 1970.

-Là ngôn ngữ được chính thức hóa vào năm 1988 bởi Viện tiêu chuẩn quốc gia Hoa Kì (ANSI).

-UNIX OS hoàn toàn được viết bằng C.

-Ngày nay C được sử dụng rộng rãi nhất và là hệ thống ngôn ngữ lập trình nổi tiếng nhất.

-Hầu hết các phần mềm hiện đại nhất được thực hiện bởi C.

**Chương trình C**

Một chương trình C có thể thay đổi từ 3 dòng đến hàng triệu dòng và nó phải được viết thành một hoặc nhiều tệp văn bản với phần nối “.c”;ví dụ, hello.c. Có thể sử dụng “vi”, “vim” hoặc bất kỳ trình soạn thảo văn bản nào khác để viết chương trình C của bạn thành một tệp.

**Các ứng dụng của lập trình C**

C ban đầu được sử dụng cho cộng việc phát triển hệ thống, đặc biệt là các chương trình tạo nên hệ điều hành. C đã được chấp nhận như một ngôn ngữ phát triển hệ thống bởi vì nó cung cấp code chạy được nhanh như code được viết bởi nhưng ngôn ngữ khác. Mội vài ví dụ về việc sử dụng C là:

-Các hệ điều hành.

-Trình biên dịch ngôn ngữ.

-Chương trình dịch hợp ngữ.

-Trình chỉnh sửa văn bản.

-Print Spoolers.

-Trình điều khiển mạng.

-Chương trình hiện đại.

-Cơ sở dữ liệu.

-Phiên dịch ngôn ngữ.

-Tiện ích.

**Về người học**

Hướng dẫn này được thiết kế cho các lập trình viên phần mềm có nhu cầu hiểu ngôn ngữ lập trình C từ đầu. Hướng dẫn về C này sẽ cung cấp cho bạn đủ hiểu biết về ngôn ngữ lập trình C từ đó bạn có thể đưa mình lên trình độ chuyên môn cao hơn.

**Điều kiện**

Trước khi tiếp tục hướng dẫn này, bạn nên hiểu cơ bản về các thuật ngữ Lập trình Máy tính. Hiểu biết cơ bản về bất kỳ ngôn ngữ lập trình nào sẽ giúp bạn hiểu các khái niệm lập trình C và tiến nhanh trên con đường học tập.

**Trình soạn thảo văn bản**

Thứ này sẽ được sử dụng để nhập chương trình của bạn. Ví dụ về một số trình chỉnh sửa bao gồm Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim hoặc vi.

Tên và phiên bản của trình soạn thảo văn bản có thể khác nhau trên các hệ điều hành khác nhau. Ví dụ: Notepad sẽ được sử dụng trên Windows và vim hoặc vi có thể được sử dụng trên windows cũng như trên Linux hoặc UNIX.

Các tệp bạn tạo bằng trình chỉnh sửa của mình được gọi là tệp nguồn và chúng chứa mã nguồn chương trình. Các tệp nguồn cho chương trình C thường được đặt tên với phần nối là ".c".

Trước khi bắt đầu lập trình, hãy đảm bảo rằng bạn đã có sẵn một trình soạn thảo văn bản và bạn có đủ kinh nghiệm để viết một chương trình máy tính, lưu nó vào một tệp, biên dịch và cuối cùng là thực thi nó.

**Trình biên dịch C**

Mã nguồn được viết trong tệp nguồn là ngôn ngữ con người có thể đọc được. Nó cần được "biên dịch", sang ngôn ngữ máy để CPU của bạn có thể thực thi chương trình theo hướng dẫn được đưa ra.

Trình biên dịch biên dịch các mã nguồn thành các chương trình thực thi cuối cùng. Trình biên dịch có sẵn miễn phí và được sử dụng thường xuyên nhất là trình biên dịch GNU C / C ++, nếu không, bạn có thể có các trình biên dịch từ HP hoặc Solaris nếu bạn có hệ điều hành tương ứng.

Phần sau giải thích về cách cài đặt trình biên dịch GNU C / C ++ trên các hệ điều hành khác nhau. Chúng ta tiếp tục đề cập đến C / C ++ cùng nhau vì trình biên dịch gcc GNU hoạt động cho cả ngôn ngữ lập trình C và C ++.

**Cài đặt trên UNIX / Linux**

Nếu bạn đang sử dụng UNIX hay Linux, sau đó kiểm tra xem GCC đã được cài đặt trên hệ thống của bạn hay chưa bằng cách nhập lệnh sau từ dòng lệnh:

$ gcc -v

Nếu bạn đã cài đặt trình biên dịch GNU trên máy của mình, thì nó sẽ in ra một thông báo như 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 GCC chưa được cài đặt, thì bạn sẽ phải tự cài đặt nó bằng cách sử dụng hướng dẫn chi tiết tại <https://gcc.gnu.org/install/>

Hướng dẫn này được viết dựa trên Linux và tất cả các ví dụ đã cho đã được biên dịch trên phiên bản Cent OS của hệ thống Linux.

**Cài đặt trên Mac OS**

Nếu bạn sử dụng Mac OS X, cách dễ nhất để có GCC là tải xuống môi trường phát triển Xcode từ trang web của Apple và làm theo hướng dẫn cài đặt đơn giản. Khi bạn đã thiết lập Xcode, bạn sẽ có thể sử dụng trình biên dịch GNU cho C / C ++.

Nếu bạn sử dụng Mac OS X, cách dễ nhất để có GCC là tải xuống môi trường phát triển Xcode từ trang web của Apple và làm theo hướng dẫn cài đặt đơn giản. Khi bạn đã thiết lập Xcode, bạn sẽ có thể sử dụng trình biên dịch GNU cho C / C ++.

Xcode hiện có sẵn tại [developer.apple.com/technologies/tools/](https://developer.apple.com/technologies/tools/).

**Cài đặt trên Windows**

Để cài đặt GCC trên Windows, bạn cần cài đặt MinGW. Để cài đặt MinGW, truy cập trang chủ MinGW,  [www.mingw.org](http://www.mingw.org/), và theo liên kết đến trang tải xuống MinGW. Tải xuống phiên bản mới nhất của chương trình MinGW, được đặt tên là MinGW- <version> .exe.

Trong khi cài đặt Min GW, tối thiểu, bạn phải cài đặt gcc-core, gcc-g ++, binutils và MinGW runtime, bạn có thể cài đặt them nếu muốn.

Thêm thư mục con bin của cài đặt MinGW vào biến môi trường PATH để bạn có thể chỉ định các công cụ này trên dòng lệnh bằng tên đơn giản của chúng.

Sau khi cài đặt xong, bạn sẽ có thể chạy gcc, g ++, ar, ranlib, dlltool và một số công cụ GNU khác từ dòng lệnh Windows.

C-Chuỗi

Chuỗi thực sự là mảng một chiều các ký tự được kết thúc bằng ký tự rỗng '\ 0'. Do đó, một chuỗi được kết thúc bằng null chứa các ký tự bao gồm chuỗi theo sau là một null.

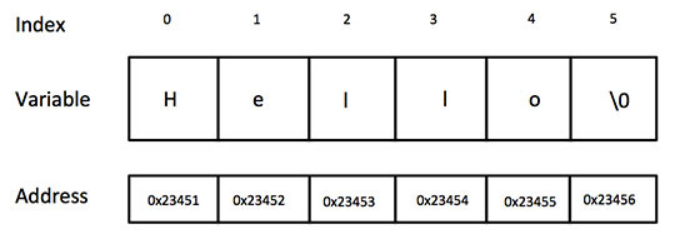
Khai báo và khởi tạo sau tạo một chuỗi bao gồm từ "Hello". Để giữ ký tự rỗng ở cuối mảng, kích thước của mảng ký tự chứa chuỗi lớn hơn một lần số ký tự trong từ "Hello"

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

Nếu bạn tuân theo quy tắc khởi tạo mảng thì bạn có thể viết câu lệnh trên như sau:

char greeting[] = "Hello";

Sau đây là phần trình bày bộ nhớ của chuỗi được định nghĩa ở trên trong C / C ++



Trên thực tế, bạn không đặt ký tự null ở cuối một hằng chuỗi. Trình biên dịch C tự động đặt '\ 0' vào cuối chuỗi khi nó khởi tạo mảng. Hãy để chúng tôi in chuỗi được đề cập ở trên –

[Live Demo](http://tpcg.io/P0muN5)

#include <stdio.h>

int main () {

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

printf("Greeting message: %s\n", greeting );

return 0;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

Greeting message: Hello

C hỗ trợ một loạt các hàm điều khiển các chuỗi bị kết thúc bằng rỗng –

|  |  |
| --- | --- |
| **Sr.No.** | **Chức năng & Mục đích** |
| 1 | **strcpy(s1, s2);**  Sao chép chuỗi s2 thành 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 giống nhau; nhỏ hơn 0 nếu s1 <s2; lớn hơn 0 nếu s1> s2. |
| 5 | **strchr(s1, ch);**  Trả về một con trỏ đến lần xuất hiện đầu tiên của ký tự ch trong chuỗi s1. |
| 6 | **strstr(s1, s2);**  Trả về một con trỏ đến lần xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1. |

Ví dụ sau sử dụng một số hàm được đề cập ở trên:

#include <stdio.h>

#include <string.h>

int main () {

char str1[12] = "Hello";

char str2[12] = "World";

char str3[12];

int len ;

/\* copy str1 into str3 \*/

strcpy(str3, str1);

printf("strcpy( str3, str1) : %s\n", str3 );

/\* concatenates str1 and str2 \*/

strcat( str1, str2);

printf("strcat( str1, str2): %s\n", str1 );

/\* total lenghth of str1 after concatenation \*/

len = strlen(str1);

printf("strlen(str1) : %d\n", len );

return 0;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

strcpy( str3, str1) : Hello

strcat( str1, str2): HelloWorld

strlen(str1) : 10

C-Cấu trúc

Mảng cho phép xác định loại biến có thể chứa một số mục dữ liệu cùng loại. Tương tự cấu trúc là một kiểu dữ liệu do người dùng định nghĩa khác có sẵn trong C cho phép kết hợp các mục dữ liệu thuộc các loại khác nhau

Các cấu trúc được sử dụng để biểu diễn một bản ghi. Giả sử bạn muốn theo dõi sách của mình trong thư viện. Bạn có thể muốn theo dõi các thuộc tính sau về mỗi cuốn sách –

* Tiêu đề
* Tác giả
* Môn học
* ID sách

Xác định cấu trúc

Để xác định một cấu trúc, bạn phải sử dụng câu lệnh struct. Câu lệnh struct định nghĩa một kiểu dữ liệu mới, với nhiều hơn một thành viên. Định dạng của câu lệnh struct như sau:

struct [structure tag] {

member definition;

member definition;

...

member definition;

} [one or more structure variables];

Nhãn cấu trúc là tùy chọn và mỗi định nghĩa thành viên là một định nghĩa biến bình thường, chẳng hạn như int i; hoặc float f; hoặc bất kỳ định nghĩa biến hợp lệ nào khác. Ở cuối định nghĩa của cấu trúc, trước dấu chấm phẩy cuối cùng, bạn có thể chỉ định một hoặc nhiều biến cấu trúc nhưng nó là tùy chọn. Đây là cách bạn khai báo cấu trúc Book –

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id;

} book;

Truy cập thành viên cấu trúc

Để truy cập bất kỳ thành viên nào của một cấu trúc, chúng tôi sử dụng toán tử truy cập thành viên (.). Toán tử truy cập thành viên được mã hóa là dấu chấm giữa tên biến cấu trúc và thành viên cấu trúc mà chúng ta muốn truy cập. Bạn sẽ sử dụng từ khóa struct để xác định các biến của kiểu cấu trúc. Ví dụ sau đây cho thấy cách sử dụng một cấu trúc trong một chương trình:

#include <stdio.h>

#include <string.h>

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id;

};

int main( ) {

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info \*/

printf( "Book 1 title : %s\n", Book1.title);

printf( "Book 1 author : %s\n", Book1.author);

printf( "Book 1 subject : %s\n", Book1.subject);

printf( "Book 1 book\_id : %d\n", Book1.book\_id);

/\* print Book2 info \*/

printf( "Book 2 title : %s\n", Book2.title);

printf( "Book 2 author : %s\n", Book2.author);

printf( "Book 2 subject : %s\n", Book2.subject);

printf( "Book 2 book\_id : %d\n", Book2.book\_id);

return 0;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book\_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book\_id : 6495700

Cấu trúc dưới dạng đối số hàm

Bạn có thể truyền một cấu trúc dưới dạng đối số hàm theo cách giống như cách bạn truyền bất kỳ biến hoặc con trỏ nào khác.

#include <stdio.h>

#include <string.h>

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id;

};

/\* function declaration \*/

void printBook( struct Books book );

int main( ) {

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info \*/

printBook( Book1 );

/\* Print Book2 info \*/

printBook( Book2 );

return 0;

}

void printBook( struct Books book ) {

printf( "Book title : %s\n", book.title);

printf( "Book author : %s\n", book.author);

printf( "Book subject : %s\n", book.subject);

printf( "Book book\_id : %d\n", book.book\_id);

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

Book title : C Programming

Book author : Nuha Ali

Book subject : C Programming Tutorial

Book book\_id : 6495407

Book title : Telecom Billing

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book\_id : 6495700

Con trỏ đến cấu trúc

Bạn có thể xác định con trỏ đến cấu trúc giống như cách bạn xác định con trỏ cho bất kỳ biến nào khác –

struct Books \*struct\_pointer;

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

struct\_pointer = &Book1;

Để truy cập các thành viên của một cấu trúc bằng cách sử dụng một con trỏ tới cấu trúc đó, bạn phải sử dụng toán tử → như sau:

struct\_pointer->title;

Chúng ta hãy viết lại ví dụ trên bằng cách sử dụng con trỏ cấu trúc.

[Live Demo](http://tpcg.io/WOoGiV)

#include <stdio.h>

#include <string.h>

struct Books {

char title[50];

char author[50];

char subject[100];

int book\_id;

};

/\* function declaration \*/

void printBook( struct Books \*book );

int main( ) {

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info by passing address of Book1 \*/

printBook( &Book1 );

/\* print Book2 info by passing address of Book2 \*/

printBook( &Book2 );

return 0;

}

void printBook( struct Books \*book ) {

printf( "Book title : %s\n", book->title);

printf( "Book author : %s\n", book->author);

printf( "Book subject : %s\n", book->subject);

printf( "Book book\_id : %d\n", book->book\_id);

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

Book title : C Programming

Book author : Nuha Ali

Book subject : C Programming Tutorial

Book book\_id : 6495407

Book title : Telecom Billing

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book\_id : 6495700

Trường bit

Trường Bit cho phép đóng gói dữ liệu trong một cấu trúc. Điều này đặc biệt hữu ích khi bộ nhớ hoặc lưu trữ dữ liệu ở mức cao. Các ví dụ điển hình bao gồm –

* Đóng gói một số đối tượng thành một từ máy ví dụ như các cờ 1 bit có thể được nén chặt
* Đọc các định dạng tệp bên ngoài - các định dạng tệp không chuẩn có thể được đọc trong ví dụ như số nguyên 9 bit

C cho phép chúng ta thực hiện điều này trong một định nghĩa cấu trúc bằng cách đặt: độ dài bit sau biến. Ví dụ -

struct packed\_struct {

unsigned int f1:1;

unsigned int f2:1;

unsigned int f3:1;

unsigned int f4:1;

unsigned int type:4;

unsigned int my\_int:9;

} pack;

Ở đây, pack\_struct chứa 6 thành viên: Bốn cờ 1 bit f1..f3, loại 4 bit và my\_int 9 bit.

C tự động đóng gói các trường bit trên càng nhỏ gọn càng tốt, với điều kiện độ dài tối đa của trường nhỏ hơn hoặc bằng độ dài từ nguyên của máy tính. Nếu không đúng như vậy, thì một số trình biên dịch có thể cho phép chồng chéo bộ nhớ cho các trường trong khi những trình biên dịch khác sẽ lưu trữ trường tiếp theo trong từ tiếp theo.

C-Mảng

Mảng là một loại cấu trúc dữ liệu có thể lưu trữ một tập hợp tuần tự có kích thước cố định của các phần tử cùng kiểu. Mảng được sử dụng để lưu trữ một tập hợp dữ liệu, nhưng thường hữu ích hơn nếu coi một mảng như một tập hợp các biến cùng kiểu

Thay vì khai báo các biến riêng lẻ, chẳng hạn như number0, number1, ... và number99, bạn khai báo một biến mảng chẳng hạn như số và sử dụng số [0], số [1] và ..., số [99] để biểu diễn các biến riêng lẻ. Một phần tử cụ thể trong một mảng được truy cập bởi một chỉ mục

Tất cả các mảng bao gồm các vị trí bộ nhớ liền kề. Địa chỉ thấp nhất tương ứng với phần tử đầu tiên và địa chỉ cao nhất cho phần tử cuối cùng



Khai báo Mảng

Để khai báo một mảng trong C, một lập trình viên chỉ định kiểu của các phần tử và số phần tử theo yêu cầu của một mảng như sau:

type arrayName [ arraySize ];

Đây được gọi là mảng một chiều. Kích thước mảng phải là một hằng số nguyên lớn hơn 0 và kiểu có thể là bất kỳ kiểu dữ liệu C hợp lệ nào. Ví dụ, để khai báo một mảng 10 phần tử được gọi là số dư kiểu double, hãy sử dụng câu lệnh này:

double balance[10];

Ở đây số dư là một mảng biến đủ để chứa tối đa 10 số kép

Khởi tạo Mảng

Bạn có thể khởi tạo một mảng trong C hoặc từng cái một hoặc sử dụng một câu lệnh như sau:

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Số lượng giá trị giữa dấu ngoặc nhọn {} không được lớn hơn số phần tử mà chúng ta khai báo cho mảng giữa dấu ngoặc vuông [].

Nếu bạn bỏ qua kích thước của mảng, một mảng vừa đủ lớn để chứa quá trình khởi tạo sẽ được tạo. Do đó, nếu bạn viết –

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

Bạn sẽ tạo chính xác mảng giống như bạn đã làm trong ví dụ trước. Sau đây là một ví dụ để gán một phần tử duy nhất của mảng:

balance[4] = 50.0;

Câu lệnh trên gán phần tử thứ 5 trong mảng với giá trị là 50.0. Tất cả các mảng đều có 0 là chỉ số của phần tử đầu tiên của chúng, còn được gọi là chỉ số cơ sở và chỉ số cuối cùng của một mảng sẽ là tổng kích thước của mảng trừ đi 1. Dưới đây là biểu diễn bằng hình ảnh của mảng mà chúng ta đã thảo luận ở trên –



Truy cập các phần tử mảng

Một phần tử được truy cập bằng cách lập chỉ mục tên mảng. Điều này được thực hiện bằng cách đặt chỉ số của phần tử trong dấu ngoặc vuông sau tên của mảng. Ví dụ -

double salary = balance[9];

Câu lệnh trên sẽ lấy phần tử thứ 10 từ mảng và gán giá trị cho biến lương. Ví dụ sau Cho thấy cách sử dụng tất cả ba khái niệm nêu trên viz. khai báo, gán và truy cập mảng –

#include <stdio.h>

int main () {

int n[ 10 ]; /\* n is an array of 10 integers \*/

int i,j;

/\* initialize elements of array n to 0 \*/

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

n[ i ] = i + 100; /\* set element at location i to i + 100 \*/

}

/\* output each array element's value \*/

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

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

}

return 0;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

Mảng chi tiết

Mảng rất quan trọng đối với C và cần được chú ý nhiều hơn. Các khái niệm quan trọng sau đây liên quan đến mảng phải rõ ràng đối với một lập trình viên C:

|  |  |
| --- | --- |
| **Stt** | **Khái niệm & Mô tả** |
| 1 | Mảng đa chiều  C hỗ trợ mảng nhiều chiều. Dạng đơn giản nhất của mảng nhiều chiều là mảng hai chiều. |
| 2 | Truyền mảng cho các hàm  Bạn có thể chuyển cho hàm một con trỏ tới một mảng bằng cách chỉ định tên của mảng mà không có chỉ mục. |
| 3 | [Trả về mảng từ một hàm](https://www.tutorialspoint.com/cprogramming/c_return_arrays_from_function.htm)  C allows a function to return an array. |
| 4 | [Con trỏ đến một mảng](https://www.tutorialspoint.com/cprogramming/c_pointer_to_an_array.htm)  Bạn có thể tạo một con trỏ tới phần tử đầu tiên của mảng bằng cách chỉ định tên mảng mà không cần bất kỳ chỉ mục nào. |

C-Quy tắc phạm vi

Phạm vi trong bất kỳ chương trình nào là một vùng của chương trình nơi một biến xác định có thể tồn tại và ngoài ngoài phạm vi của biến đó, nó không thể được truy cập. Có ba nơi mà các biến có thể được khai báo trong ngôn ngữ lập trình C –

* Bên trong một hàm hoặc một khối được gọi là các biến cục bộ
* Bên ngoài tất cả các hàm được gọi là biến toàn cục
* Trong định nghĩa của các tham số hàm được gọi là tham số hình thức

Hãy để chúng tôi hiểu các biến cục bộ và toàn cục, và các tham số chính thức là gì

Biến cục bộ

Các biến được khai báo bên trong một hàm hoặc khối được gọi là biến cục bộ. Chúng chỉ có thể được sử dụng bởi các câu lệnh bên trong hàm hoặc khối mã đó. Các biến cục bộ không được biết đến với các hàm bên ngoài của chúng. Ví dụ sau đây cho thấy cách các biến cục bộ được sử dụng. Ở đây tất cả các biến a, b và c là cục bộ của hàm main ()

#include <stdio.h>

int main () {

/\* local variable declaration \*/

int a, b;

int c;

/\* actual initialization \*/

a = 10;

b = 20;

c = a + b;

printf ("value of a = %d, b = %d and c = %d\n", a, b, c);

return 0;

}

Biến toàn cục

Các biến toàn cục được định nghĩa bên ngoài một hàm, thường nằm trên đầu chương trình. Các biến toàn cục giữ các giá trị của chúng trong suốt thời gian tồn tại của chương trình của bạn và chúng có thể được truy cập bên trong bất kỳ hàm nào được định nghĩa cho chương trình. Một biến toàn cục có thể được truy cập bởi bất kỳ hàm nào. Tức là, một biến toàn cục có sẵn để sử dụng trong toàn bộ chương trình của bạn sau khi khai báo. Chương trình sau đây cho thấy cách các biến toàn cục được sử dụng trong một chương trình

#include <stdio.h>

/\* global variable declaration \*/

int g;

int main () {

/\* local variable declaration \*/

int a, b;

/\* actual initialization \*/

a = 10;

b = 20;

g = a + b;

printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

return 0;

}

Một chương trình có thể có cùng tên cho các biến cục bộ và toàn cục nhưng giá trị của biến cục bộ bên trong một hàm sẽ được ưu tiên hơn. Đây là một ví dụ -

#include <stdio.h>

/\* global variable declaration \*/

int g = 20;

int main () {

/\* local variable declaration \*/

int g = 10;

printf ("value of g = %d\n", g);

return 0;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

value of g = 10

Thông số chính thức

Các tham số chính thức, được coi là các biến cục bộ có trong một hàm và chúng được ưu tiên hơn các biến toàn cục. Sau đây là một ví dụ -

#include <stdio.h>

/\* global variable declaration \*/

int a = 20;

int main () {

/\* local variable declaration in main function \*/

int a = 10;

int b = 20;

int c = 0;

printf ("value of a in main() = %d\n", a);

c = sum( a, b);

printf ("value of c in main() = %d\n", c);

return 0;

}

/\* function to add two integers \*/

int sum(int a, int b) {

printf ("value of a in sum() = %d\n", a);

printf ("value of b in sum() = %d\n", b);

return a + b;

}

Khi đoạn mã trên được biên dịch và thực thi, nó tạo ra kết quả sau:

value of a in main() = 10

value of a in sum() = 10

value of b in sum() = 20

value of c in main() = 30

Khởi tạo các biến cục bộ và toàn cục

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

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

Thực hành lập trình tốt là khởi tạo các biến đúng cách, nếu không chương trình của bạn có thể tạo ra kết quả không mong muốn, vì các biến chưa được khởi tạo sẽ nhận một số giá trị rác đã có sẵn tại vị trí bộ nhớ của chúng.

C – Hàm(s)

Hàm là 1 tập hợp các câu lệnh với nhau để thực hiện một nhiệm vụ . Mỗi chương trình C phải có ít nhất 1 câu lệnh là **main()** và tất cả chương trình tầm thường có thể định nghĩa hàm thêm vào

Bạn có thể chia code của bạn vào các hàm riêng biệt . Cách bạn chia code của bạn giữa các hàm khác nhau là tùy thuộc vào bạn , nhưng nên phân chia 1 cách logic để mỗi hàm đảm nhận 1 nhiệm vụ cụ thể

Khai báo hàm nói với trình biên dịch về tên của hàm , kiểu trả về , tham số . Một hàm định nghĩa cung cấp phần thân thực sự của hàm

Thư viện tiêu chuẩn của C cung cấp nhiều hàm có sẵn để chương trình của bạn có thể gọi . Ví dụ như strcat() để ghép 2 chuỗi lại , memcpy() để sao chép 1 địa chỉ bộ nhớ này sang 1 địa chỉ bộ nhớ khác , và còn nhiều hàm nữa

Một hàm cũng có thể gọi như một hệ thống hoặc 1 chương trình con hoặc 1 câu lệnh vân..vân..mây..mây…

Định nghĩa 1 hàm

Hình thức chung của 1 định nghĩa hàm trong ngôn ngữ lập trình C là như sau :

return\_type function\_name( parameter list ) {

body of the function

}

Một định nghĩa hàm trong ngôn ngữ C bao gồm một tiêu đề hàm và một thân hàm .

Đây là tất cả các phần trong một hàm –

* Kiểu trả về - Một hàm có thể trả về một giá trị . Kiểu trả về là kiểu dữ liệu của giá trị mà hàm trả về . 1 số hàm thực hiện phép toán yêu cầu mà không trả về giá trị . Trong trường hợp này , kiểu trả về có từ khóa là void
* Tên hàm – Đây là tên thực sự của hàm . Tên hàm và danh sách biến cùng nhau tạo thành 1 hàm chữ ký ( hàm do bạn tự tạo :> )
* Biến – 1 biến giống 1 chỗ giữ đồ . Khi 1 hàm được gọi , bạn phải truyền vào giá trị cho 1 biến . Giá trị này được gọi như 1 biến thật sự hoặc đối số . Giá trị được chuyển đến kiểu , thứ tự , và số lượng của biến trong hàm . Biến thì không bắt buộc phải có , một hàm có thể không cần chứa biến
* Thân hàm – Thân hàm chứa một tập hợp các câu lệnh để định nghĩa hàm phải làm gì

Ví dụ:

Cho bên dưới là 1 mã nguồn cho 1 hàm gọi là max() . Hàm lấy 2 biến num1 và num2 và trả về giá trị lớn nhất giữa 2 biến

/\* function returning the max between two numbers \*/

int max(int num1, int num2) {

/\* local variable declaration \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

Khai báo hàm

Khai báo 1 hàm nói lên trình biên dịch về tên hàm và cách gọi hàm . Phần thân thực của hàm có thể được định nghĩa riêng biệt.

Một khai báo hàm có các phần sau

return\_type function\_name( parameter list );

Đối với hàm max () được định nghĩa ở trên, khai báo hàm như sau:

int max(int num1, int num2);

Tên tham số không quan trọng trong khai báo hàm chỉ cần kiểu của chúng, vì vậy sau đây cũng là một khai báo hợp lệ:

int max(int, int);

Khai báo hàm là bắt buộc khi bạn xác định một hàm trong một tệp nguồn và bạn gọi hàm đó trong một tệp khác. Trong trường hợp này, bạn nên khai báo hàm ở đầu tệp gọi hàm.

Gọi một hàm

Trong khi tạo một hàm C, bạn đưa ra định nghĩa về những gì hàm phải làm. Để sử dụng một hàm, bạn sẽ phải gọi hàm đó để thực hiện tác vụ đã xác định

Khi một chương trình gọi một hàm, điều khiển chương trình được chuyển đến hàm được gọi. Một hàm được gọi thực hiện một tác vụ được xác định và khi câu lệnh trả về của nó được thực thi hoặc khi đạt đến dấu ngoặc nhọn kết thúc hàm, nó sẽ trả lại điều khiển chương trình trở lại chương trình chính

Để gọi một hàm, bạn chỉ cần chuyển các tham số bắt buộc cùng với tên hàm và nếu hàm trả về một giá trị, thì bạn có thể lưu trữ giá trị trả về. Ví dụ -

#include <stdio.h>

/\* function declaration \*/

int max(int num1, int num2);

int main () {

/\* local variable definition \*/

int a = 100;

int b = 200;

int ret;

/\* calling a function to get max value \*/

ret = max(a, b);

printf( "Max value is : %d\n", ret );

return 0;

}

/\* function returning the max between two numbers \*/

int max(int num1, int num2) {

/\* local variable declaration \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

Chúng tôi đã giữ max () cùng với main () và biên dịch mã nguồn. Trong khi chạy tệp thực thi cuối cùng, nó sẽ tạo ra kết quả sau:

Max value is : 200

Đối số hàm

Nếu một hàm sử dụng các đối số, nó phải khai báo các biến chấp nhận giá trị của các đối số. Các biến này được gọi là tham số chính thức của hàm

Các tham số chính thức hoạt động giống như các biến cục bộ khác bên trong hàm và được tạo khi nhập vào hàm và bị hủy khi thoát

Trong khi gọi một hàm, có hai cách mà các đối số có thể được truyền cho một hàm –

|  |  |
| --- | --- |
| **Stt.** | **Gọi loại và sự miêu tả** |
| 1 | Gọi bởi giá trị  Phương thức này sao chép giá trị thực của một đối số vào tham số chính thức của hàm. Trong trường hợp này, các thay đổi được thực hiện đối với tham số bên trong hàm không ảnh hưởng đến đối số. |
| 2 | Gọi bởi tham chiếu  Phương thức này sao chép địa chỉ của một đối số vào tham số chính thức. Bên trong hàm, địa chỉ đã từng được sử dụng để truy cập đối số thực được sử dụng trong lệnh gọi. Điều này có nghĩa là các thay đổi được thực hiện đối với tham số ảnh hưởng đến đối số. |

Theo mặc định, C sử dụng lệnh gọi theo giá trị để chuyển các đối số. Thông thường , nó có nghĩa là mã bên trong một hàm không thể thay đổi các đối số được sử dụng để gọi hàm.