\*Con trỏ (Pointer)

Địa chỉ trong C

* Nếu bạn có một biến var trong chương trình, &var sẽ trả về chính địa chỉ của nó trong bộ nhớ.
* Chúng ta đã sử dụng địa địa chỉ nhiều lần trong khi đang sử dụng hàm scanf ().



1. Khái niệm

* Con trỏ là một biến, nó chứa địa chỉ của vị trí bộ nhớ của biến khác.
* Nếu một biến chứa địa chỉ của biến khác, biến đầu tiên được gọi là địa chỉ của biến thứ hai.
* Một con trỏ cung cấp một phương thức gián tiếp để truy cập vào giá trị của một mục dữ liệu.
* Con trỏ có thể trỏ đến biến của các kiểu dữ liệu cơ bản khác như **int***,* **char** hoặc **double** hoặc dữ liệu tổng hợp như mảng (**array**) hoặc **structures** (kiểu dữ liệu do người dùng tự định nghĩa).

1. Biến con trỏ

* Khai báo một con trỏ bao gồm kiểu cơ sở và tên biến đặt ngay sau ký hiệu (\*).

Ex: int \*ptr;

1. Toán tử con trỏ

* Có 2 toán tử đặc biệt đó là sử dụng với con trỏ **&** và **\***.
* Toán tử **&** là toán tử một ngôi và nó trả về địa chỉ bộ nhớ của toán hạng

Ex: ptr = &var;

* Toán tử **\*** là sự bổ sung của **&**, trả về giá trị được chứa trong vị trí bộ nhớ được trỏ đến bởi giá trị của biến con trỏ.

Ex: value = \*ptr;

1. Con trỏ số học

* Cộng và trừ là các hoạt động duy nhất có thể được thực hiện trên con trỏ.

Ex: int var, \*ptr;

ptr = &var;

var = 100;

ptr++;

* Mỗi lần con trỏ được tăng lên, nó trỏ đến vị trị bộ nhớ của phần tử tiếp theo hoặc kiểu cơ sở của nó.
* Mỗi lần con trỏ bị giảm xuống, nó trỏ đến vị trí của phần tử trước đó.
* Tất cả các con trỏ khác sẽ tăng lên hoặc giảm xuống phụ thuộc vào độ dài của kiểu dữ liệu mà chúng đang trỏ tới.

|  |  |
| --- | --- |
| **Con trỏ số học** | **Mô tả** |
| ++ptr\_var hoặc ptr\_var++ | Trỏ đến số nguyên tiếp theo sau var |
| --ptr\_var hoặc ptr\_var-- | Trỏ đến số nguyên trước var |
| ptr\_var + 1 | Trỏ đến vị trí thứ 1 sau var |
| ptr\_var - 1 | Trỏ đến vị trí thứ 1 trước var |
| ++\*ptr\_var hoặc (\*ptr\_var)++ | Tăng var 1 đơn vị |
| \*ptr\_var++ | Sẽ lấy giá trị của số nguyên tiếp theo sau var |

1. So sánh con trỏ

* Hai con trỏ có thể được so sánh trong một biểu thức quan hệ cung cấp hai con trỏ đang trỏ đến biến có cùng kiểu dữ liệu.
* Những con trỏ có cùng kiểu dữ liệu (sau khi chuyển đổi con trỏ) có thể được so sánh về sự công bằng. Hai con trỏ có cùng kiểu dữ liệu so sánh bằng nhau khi và chỉ khi chúng đều NULL, cả 2 đều trỏ đến hàm giống nhau hoặc đại diện cho địa chỉ giống nhau.
* Xem xét rằng **ptr\_a** và **ptr\_b** là hai biến con trỏ, chúng trỏ đến các phần tử dữ liệu a và b.

1. Con trỏ NULL

* Nó luôn luôn là một thực tế tốt để gán một giá trị NULL đến biến con trỏ trong trường hợp bạn không có một địa chỉ chính xác để gán. Điều này được thực hiện tại thời điểm khai báo biến.
* Một con trỏ được gán NULL thì được gọi là con trỏ không có giá trị.
* Con trỏ NULL là một hằng số với giá trị định nghĩa bằng 0 trong vài thư viện tiêu chuẩn.

Ex: int \*ptr = NULL;

1. Con trỏ và mảng một chiều

* Trong hầu hết ngữ cảnh, tên của mảng phân rã thành con trỏ. Để dễ hiểu, tên mảng được chuyển đổi thành con trỏ và nó trỏ đến phần tử đầu tiên trong mảng đó.
* Đó là lý do tại sao bạn có thể sử dụng các con trỏ để truy cập vào các phần tử của nhiều mảng. Tuy nhiên, bạn nên nhớ rằng con trỏ và mảng là khác nhau.
* Địa chỉ của một phần tử mảng có thể được thể hiện trong 2 trường hợp:
  + Bằng cách viết phần tử mảng ngay sau dấu **&**.
  + Bằng cách viết một biểu thức trong đó chỉ số phụ được thêm vào tên mảng.

1. Con trỏ và mảng đa chiều

* Một mảng đa chiều có thể được định nghĩa như một con trỏ đến một nhóm tiếp giáp các mảng một chiều.
* Khai báo một mảng đa chiều có thể được viết:
  + Data\_type (\*ptr\_var) [expr\_2];
* Thay vì:
  + Data\_type ptr\_var [expr\_1][expr\_2];

1. Cấp phát động

* Như bạn biết, một mảng là tập hợp tuần tự các phần tử có cùng kiểu dữ liệu. Khi kích thước của một mảng được khai báo, bạn không thể thay đổi nó.
* Đôi khi kích thước của mảng do bạn khai báo có thể không đủ. Để giải quyết vấn đề này, bạn có thể tự cấp phát bộ nhớ trong suốt thời gian chạy. Đó được biết như cấp phát động trong C.
* Ngôn ngữ lập trình C cung cấp vài hàm để phân bổ và quản lý bộ nhớ.
* Các hàm này có thể được tìm tại thư viện tiền xử lý **<stdlib.h>**.
  + void \*malloc (int num);
* Đây là hàm cấp phát bộ nhớ cho một mảng và để chúng được khởi tạo.
  + void \*calloc (int num, int size);
* Đây là hàm cấp phát bộ nhớ cho một mảng các phần tử số mỗi kích thước trong byte sẽ có kích thước.
  + void \*realloc (void \*address, int newsize);
* Đây là hàm cấp phát lại bộ nhớ để mở rộng kích thước.
  + void free (void \*address);
* Đây là hàm giải phóng bộ nhớ.
* Khi ta khởi tạo một biến con trỏ và chưa gán giá trị địa chỉ nào cho nó, con trỏ sẽ tìm một vùng nhớ bất kì trên RAM để gán cho nó hoặc khi ta chưa muốn gán giá trị nào cho nó thì ta gán cho nó giá trị = **NULL**. Một cách khác là ta tự cấp phát vùng nhớ cho biến con trỏ đó và nó sẽ biến thành mảng.

1. Con trỏ hàm

* Trong C, nó cũng có thể chuyển địa chỉ dưới dạng đối số để hoạt động.
  + Gọi bởi giá trị.
  + Gọi bởi tham chiếu.
* Các con trỏ hàm trỏ đến code như những con trỏ bình thường. Trong con trỏ hàm, tên hàm có thể được sử dụng để lấy ra địa chỉ của hàm.
* Một hàm có thể cũng được thông qua một đối số và có thể trả về từ một hàm.
* Bởi khi sử dụng con trỏ hàm, một hàm có thể gửi một tham số đến hàm khác.
* Đây là tính năng cho phép trong C để tải hàm một cách tự động trong thời gian chạy.
* Cú pháp:
  + Function\_return\_type (\*pointer\_name) (function argument list)