Pointer

* Pointer base:

1. Address of operator:

* Cho phép xem địa chỉ của 1 biến, vùng nhớ.

Syntax: &<variable>;

1. Dereference operator: (toán tử trỏ đến)

* Cho phép ta truy cập, get/set giá trị tại 1 địa chỉ cụ thể

Syntax: \*.

Vd: \*&<variable>=giá trị của biến= variable.

1. Pointer:

* Là biến chứa địa chỉ(address) bộ nhớ của giá trị của nó.
* Cách khai báo:

Syntax: <datatypes> \*<variable>;

Khai báo và gán giống biến bình thường, nhưng biến ở đây chứa địa chỉ.

* Quy tắc ngầm đặt dấu:

Khi khai báo biến thì đặt dấu sát biến, còn với hàm thì đặt dấu sát kiểu dữ liệu.

Vd: biến: int \*a;

Hàm: int\* dosomething;

* Ngoài chứa địa chỉ ra thì còn lại như biến thường.
* Kiểu dữ liệu của con trỏ= kiểu dữ liệu của biến mà nó trỏ đến.

1. Notes:

* Khi khai báo biến trỏ(biến chứa địa chỉ ấy), thì phần biến nó sẽ chứa địa chỉ.

Vd: int \*pnt;

Lúc này pnt=address.

* Muốn hiện giá trị của biến trỏ(address) ta chỉ cần dùng con trỏ trỏ vào.

Pnt=address; mà \*pnt=value.

* Sức mạnh của con trỏ: Khi ta thay đổi giá trị ở địa chỉ mà biến trỏ đang trỏ đến, thì giá trị biến của vùng dữ liệu ấy cũng thay đổi theo.

Vd: int value=10;

Int \*pnt=&value;

\*pnt=100;

=> value=100;

=> Khi biết được địa chỉ ta có thể thay đổi tùy thích giá trị của vùng dữ liệu đấy.(cẩn thận bị hack ☺)

- Cuối cùng : Nếu ta cố gắng truy xuất giá trị của con trỏ khi nó chưa gán giá trị cụ thể(giá trị rác), thì chương trình có thể tự động đóng vì lý do bảo mật.

* Null pointer:

1. Chức năng, công dụng:

* Là giá trị con trỏ đặc biệt, con trỏ có giá trị đấy thì ta hiểu là biến con trỏ lúc này không chứa bất kỳ 1 địa chỉ nào cả.
* Ưu tiên khai báo biến con trỏ có giá trị null thay vì rác.

1. Syntax:

* Cách 1: Ta có thể khai báo giá trị 0 như giá trị null, nên ta sẽ có kỹ thuật khai báo : #define null 0; rồi sử dụng hằng null làm giá trị null. Tuy nhiên, đây là 1 kỹ thuật ở C, C++ tuy vẫn hiểu nhưng nó sẽ không nhất quán vì giá trị Null != 0. Vì vậy, không khuyến khích sử dụng.
* Cách 2: ở C++ 11, ta có từ khóa nullptr là biểu diễn cho gtri null, nên ta cứ sử dụng bình thường:

Int \*ptr=nullptr;

1. Notes:

* Khi cố truy cập vào giá trị null thì chương trình sẽ dễ bị chết.
* Pointer and Array:

1. Chức năng, công dụng:

* Bản chất của mảng: Mảng bản chất là 1 dãy bộ nhớ, được xác định ở phần tử đầu tiên array[0], các giá trị sau thì ta chỉ việt cộng thêm số dữ liệu của từng kiểu.

Vd: Int A[4];

Mảng này bản chất ở chỗ xác định vị trí của A[0], sau đó A[1]=A[0]+4 byte (kiểu int), A[2]=A[1]+4 byte….

* Chính vì thế, giá trị của array=&a[0]=&array.
* Con trỏ vào mảng, thực chất là việc ta sử dụng biến con trỏ để trỏ vào giá trị đầu tiên của array, từ đó ta sẽ lấy được vị trí định vị của mảng a[0], từ đó ta có thể sử dụng mảng.
* Về bản chất, con trỏ và mảng khá là giống nhau, nhưng chúng vẫn có sự phân biệt như:

+ Khi dùng mảng ta có thể xác định được số phần tử, còn con trỏ thì không.

+ Khi sử dụng mảng thì nó sẽ lấy toàn bộ bộ nhớ của tất cả phần tử trong mảng.

VD: int A[4];

Thì mảng sẽ có kích thước là 16byte

Còn int \*ptr=array;

Thì mảng sẽ có kích thước là 4byte.

* Kiểu mảng tĩnh thì nó sẽ tạo ra 1 vùng tổng bộ nhớ để chứa các phần tử trong mảng còn con trỏ vào mảng thì nó chỉ trỏ vào và lấy vùng bộ nhớ của 1 phần tử đấy thôi.
* Sử dụng con trỏ phải truyền vào địa chỉ của mảng.

1. Syntax:

Int array[];

int \*ptr=array; (trỏ vào địa chỉ a[0]).

1. Note:

* Khi truyền mảng vào hàm thì mảng sẽ ngầm định thành con trỏ, trỏ đến giá trị đầu tiên của mảng, con trỏ vào mảng cũng giống vậy.
* Pointer arithmetic:

1. Chức năng, công dụng:

* Nguyên lý: nếu ptr trỏ đến 1 số nguyên, ptr+1 là địa chỉ số nguyên tiếp theo trong bộ nhớ sau pointer. Tương tự, ngược lại với ptr-1.

VD: int \*ptr=0x00001;

* Ptr+1=> ptr=0x00004; //Cách nhau 4 byte.(kiểu int).
* Ptr trong mảng: Như ta đã biết mảng là 1 vùng dữ liệu liên tiếp nhau, được xác định bởi địa chỉ &a[0], or array. Vì vậy, nếu ta dùng ptr trỏ vào array, và dùng các phép toán, ta sẽ có 1 mảng con trỏ.

Vd: int \*ptr=array; //array là địa chỉ.

* &a[1]=ptr+1=array+1; // address
* a[1]=\*(ptr+1)=\*(array+1); // value.
* Mảng tĩnh thì ta có thể biết được số phần tử, còn ptr thì không.

1. Syntax:

* Address: int \*ptr=<array>;

Int \*ptr=&array or &array[0], &array[n];

* Value: int \*ptr=array;

\*(ptr+n) = value of array[n];

Or \*(array+n) value of array[n];

* Thêm con trỏ vào trước địa chỉ là ra giá trị.

1. Notes:

* Vì tính chất sử dụng các phép toán, nên ta hoàn toàn có thể dùng vòng lặp để duyệt các phần tử của mảng.
* Lưu ý: pointer+1 là vùng bộ nhớ tiếp theo của kiểu dữ liệu đó, không phải vùng bộ nhớ kế tiếp của vùng địa chỉ lúc này.

VD: int \*ptr;

Ptr+1=<vùng địa chỉ bộ nhớ ban đầu lúc mới khai báo>+4 byte.

Không phải là ptr đang ở khoảng 2byte, thì compiler sẽ nâng lên 3 byte, compiler sẽ nâng lên 4byte, mặc kệ là ta sử dụng bao nhiêu vùng nhớ.

* Const:
* Một con trỏ bình thường không thể trỏ vào vùng bộ nhớ của hẳng được.

1. Pointer to const value:

* Như tên nó là 1 dạng con trỏ đặc biệt, con trỏ hằng, có thể trỏ thẳng vào vùng nhớ của hằng, ta có thể thao tác trên vùng nhớ hằng bằng con trỏ này.
* Con trỏ này không thể thay đổi giá trị (value) của vùng nhớ, bù lại nó có khả năng thay đổi địa chỉ trỏ tới.

Vd: const int a=76;

Int b=54;

Int \*ptr=&a;

Ptr=&b; => có thể thay đổi địa chỉ trỏ tới.

* Khi nó trỏ tới bộ nhớ nào thì các thao tác trên bộ nhớ ấy đều được xem như hằng. (nghĩa là không thể thay đổi value của bộ nhớ.)
* Syntax: const <datatypes> \*<pointer variable>;

1. Const pointer:

* Là con trỏ có thể trỏ tới biến, not hằng.
* Con trỏ này không thể thay đổi địa chỉ, bù lại có thể thay đổi value của vùng bộ nhớ trên hằng. Vì không thể thay đổi được địa chỉ nên phải khởi tạo địa chỉ khi khai báo.

Vd:

Int a;

Int \*const ptr=&a;

Ptr= 20;

* Syntax: int \*const <pointer variable>=address;

1. Const pointer to a const value:

* Có đầy đủ tính chất của 2 ông trên, không đổi đc địa chỉ lẫn giá trị.
* Syntax: cons tint \*const <pointer variable>= address;
* Con trỏ void:

1. Không nên dùng
2. Syntax:

Void \*ptr;

* Có thể dùng để trỏ đến bất kỳ kiểu dữ liệu nào, nhưng cần phải xác định kiểu dữ liệu.(củ chuối zz)

Vd:

Int value1=15;

Void \*ptr;

Ptr=value1; (ok nhưng k truy xuất dữ liệu của con trỏ void đc).

Ta cần phải: stactic\_cast<int\*>(ptr) thì mới truy xuất dữ liệu được.

* Pointer to pointer, 2-dimensional array pointer:

1. Pointer to pointer:

* Giải thích :

Int value=10;

(giá trị ban đầu).

Int \*ptr=&value;

(con trỏ ptr trỏ vào và lấy địa chỉ của value)

Int \*\*ptr\_ptr=&ptr;

(con trỏ ptr\_ptr trỏ vào và lấy địa chỉ của con trỏ ptr)

* Con trỏ trỏ đến con trỏ không thế lấy được địa chỉ trực tiếp của biến.

1. Syntax:

Int value=int;

Int \*ptr=&value;

Int \*\*ptr\_ptr=&ptr; (……)

1. Notes:

* Ít sử dụng bớt vì nó hơi phức tạp nên dễ sai.
* Có thể dùng để khai báo mảng 2 chiều.