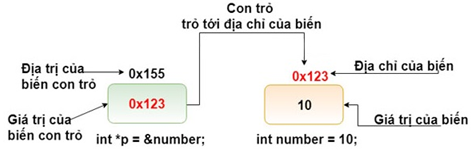
# CHƯƠNG VIII: CON TRỎ

## GIỚI THIỆU CON TRỎ

### Giới thiệu con trỏ

#### Con trỏ là gì

Con trỏ trong C là một loại biến đặc biệt dùng để lưu trữ địa chỉ của một biến khác trong bộ nhớ, mà **giá trị** của nó là **địa chỉ của 1 biến khác.**



Một số khái niệm về con trỏ:

* Giá trị của con trỏ: **địa chỉ** mà con trỏ đang trỏ đến.
* Địa chỉ của con trỏ: địa chỉ của bản thân biến con trỏ đó.
* Giá trị của biến: nơi con trỏ đang trỏ tới.
* Địa chỉ của biến nơi con trỏ đang trỏ tới = giá trị của con trỏ.

Chính vì con trỏ mang địa chỉ, nó là một biến đặc biệt có thêm những quyền năng mà biến bình thường không có. Nhờ việc nó mang địa chỉ, nó có thể trỏ lung tung trong bộ nhớ.

#### Khai báo con trỏ

Con trỏ trong C cũng có thể khai báo giống như biến bình thường, tên biến là một định danh hợp lệ.

|  |
| --- |
| **Cú pháp:** |
| <Kiểu dữ liệu> \*<Tên biến>; |

Trong đó:

* Kiểu dữ liệu có thể là: void, int, float, double,…
* Dấu **\*** trước tên biến là ký hiệu báo cho trình biên dịch biết ta đang khai báo con trỏ.

**Ví dụ:**

|  |
| --- |
| int \*p; // khai báo con trỏ để trỏ tới biến kiểu nguyên  // khai báo con trỏ p kiểu int, biến val (không phải con trỏ)  int \*p, val;  float \*p; // khai báo con trỏ để trỏ tới biến kiểu thực  char \*p; // khai báo con trỏ để trỏ tới biến kiểu ký tự  void \*p; // con trỏ kiểu void (không kiểu), đây là 1 kiểu đặc biệt của con trỏ. Con trỏ kiểu void có thể trỏ đến bất kỳ đối tượng nào (với bất kỳ kiểu dữ liệu nào) có địa chỉ cụ thể trên bộ nhớ ảo. |

**Chú ý, kiểu dữ liệu của con trỏ phải trùng với kiểu dữ liệu của biến mà con trỏ sẽ trỏ đến.**

#### Gán giá trị cho con trỏ

Sau khi khai báo con trỏ, bạn cần khởi tạo giá trị cho nó. Nếu con trỏ được sử dụng mà không được khởi tạo, giá trị của nó sẽ là giá trị rác, điều này sẽ làm chương trình của bạn chạy không đúng, thậm chí là nguy hiểm nếu giá trị rác đó chẳng may lại chính là địa chỉ của 1 biến nào đó bạn đang dùng.

**VD:** Gán giá trị cho con trỏ

|  |
| --- |
| int \*p, value;  value = 5;  p = &value; // khởi tạo giá trị cho con trỏ p là địa chỉ của value |

Hoặc bạn có thể khai báo và khởi tạo đồng thời:

|  |
| --- |
| int value = 5;  int \*p = &value; // khai báo con trỏ p và khởi tạo giá trị cho con trỏ là địa chỉ của value |

Khi khai báo con trỏ mà hiện chưa sử dụng đến thì chúng ta nên gán giá trị cho nó là NULL (phải được viết hoa). Khi đó con trỏ NULL luôn có giá trị 0.

Việc gán giá trị NULL cho một biến con trỏ luôn là một thực tiễn tốt trong trường hợp bạn không có địa chỉ chính xác để gán. Điều này được thực hiện tại thời điểm khai báo biến. Một con trỏ được gán NULL được gọi là con trỏ null .

Con trỏ NULL là một hằng số có giá trị bằng 0 được xác định trong một số thư viện chuẩn. Hãy xem xét chương trình sau:

|  |
| --- |
| **VD:** Gán giá trị của con trỏ là NULL |
| #include <stdio.h>  int main () {  int  \*p;  p = NULL;  printf("\nDia chi con tro p: %d - Gia tri cua con tro p: %d", &p, p);  return 0;  } |
| **Kết quả:** |
| Dia chi con tro p: 6422300 - Gia tri cua con tro p: 0 |

Trong hầu hết các hệ điều hành, các chương trình không được phép truy cập bộ nhớ ở địa chỉ 0 vì bộ nhớ đó được hệ điều hành dành riêng. Tuy nhiên, địa chỉ bộ nhớ 0 có ý nghĩa đặc biệt; nó báo hiệu rằng con trỏ không nhằm mục đích trỏ đến một vị trí bộ nhớ có thể truy cập được. Nhưng theo quy ước, nếu một con trỏ chứa giá trị NULL (\0), nó được coi là không trỏ đến.

Để kiểm tra con trỏ null, bạn có thể sử dụng câu lệnh 'if' như sau:

|  |
| --- |
| if(ptr) /\* Nó không phải là con trỏ NULL \*/  if(!ptr) /\* Nó là con trỏ NULL \*/ |

**Lưu ý:**

|  |
| --- |
| * Con trỏ khi khai báo nên được khởi tạo giá trị ngay. * Con trỏ kiểu **void** là loại biến con trỏ tổng quát, nó có thể nhận địa chỉ của biến bất kỳ ở bất cứ kiểu dữ liệu nào. |

Ta cần phân biệt rõ giữa **khai báo con trỏ**, **giá trị của con trỏ** và **giá trị của biến mà con trỏ đang trỏ đến.**

**Ví dụ:**

|  |
| --- |
| **// Khai báo, Khởi tạo con trỏ p có giá trị là địa chỉ biến a**  int \*p, a = 5;  p = &a;  **// Vừa khai báo và khởi tạo con trỏ p có giá trị là địa chỉ biến a**  int a = 5;  int \*p = &a;  **// (Giá trị của biến mà con trỏ p đang trỏ tới) gán = 5**  \*p = 5 |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {  int a = 5, b = 8;  int \*p;   //Khai báo con trỏ p  p = &a;   //Giá trị của con trỏ p là địa chỉ của biến a  printf("\nDia chi a: %i - Gia tri cua a: %i", &a, a);  printf("\nDia chi p: %i - Gia tri cua p: %i", &p, p);  printf("\nDia chi p: %i - Gia tri cua p: %i\n", &p, \*p);  \*p = b;  //\*p là lấy giá trị của biến mà con trỏ đang trỏ đến.  printf("\nDia chi p: %i - Gia tri cua p: %i", &p, p);  printf("\nDia chi p: %i - Gia tri cua p: %i", &p, \*p);  printf("\nDia chi a: %i - Gia tri cua a: %i", &a, a);  } |

**Kết quả:**

|  |
| --- |
| Dia chi a: 6422296 - Gia tri cua a: 5  Dia chi p: 6422292 - Gia tri cua p: 6422296  Dia chi p: 6422292 - Gia tri cua p: 5  Dia chi p: 6422292 - Gia tri cua p: 6422296  Dia chi p: 6422292 - Gia tri cua p: 8  Dia chi a: 6422296 - Gia tri cua a: 8 |

#### Bản chất của con trỏ trong C

Bạn sẽ hiểu rõ hơn các quyền năng của con trỏ trong phần này, cũng xem ví dụ dưới đây nào:

|  |
| --- |
| #include <stdio.h>  int main() {  **// Khai báo + khởi tạo biến value = 10**  int value = 10;  **// Lấy giá trị của biến value**  printf("\nGia tri cua \"value\" = %i", value);  **// Lấy địa chỉ của biến value**  printf("\nDia tri cua \"value\" = %i", &value);  printf("\n-------------------\n");    **/\* Khai báo + khởi tạo biến con trỏ p có giá trị là địa chỉ của biến value \*/**  int \*p = &value;  **// Lấy giá trị của con trỏ p**  printf("\nGia tri cua con tro \"p\" = %i", p);  **// Lấy địa chỉ của con trỏ p**  printf("\nDia tri cua con tro \"p\"  = %i", &p);  **// Lấy giá trị của biến mà con trỏ p đang trỏ tới dùng toán tử \***  printf("\nGia tri cua bien ma con tro \"p\" dang tro toi = %i", \*p);  printf("\n-------------------\n");    **/\*Thay đổi giá trị của biến value thông qua con trỏ p. Giống như hàm scanf() có thể thay đổi giá trị của biến khi nhận vào địa chỉ, con trỏ khi có địa chỉ của 1 biến hoàn toàn có thể thay đổi giá trị của**  **biến đó theo cách dưới đây:\*/**  **// Lấy giá trị của biến value**  printf("\nGia tri cua \"value\" = %i", value);  **// Thay đổi giá trị của biến value thông qua con trỏ p**  \*p = 100;  **// Lấy giá trị của biến value**  printf("\nGia tri cua \"value\" = %i", value);  **// Lấy giá trị của biến mà con trỏ p đang trỏ tới dùng toán tử \***  printf("\nGia tri cua bien ma con tro \"p\" dang tro toi = %i", \*p);  printf("\n-------------------\n");    **/\*Việc lấy giá trị của biến thông qua con trỏ chỉ là 1 cách khác để lấy được giá trị của biến đó.\*/**  value = 1000;  **// Lấy giá trị của biến value**  printf("\nGia tri cua \"value\" = %i", value);  **// Lấy giá trị của biến mà con trỏ p đang trỏ tới dùng toán tử \***  printf("\nGia tri cua bien ma con tro \"p\" dang tro toi = %i", \*p);  } |

**Kết quả:**

|  |
| --- |
| Gia tri cua "value" = 10  Dia tri cua "value" = 6487580  -------------------  Gia tri cua con tro "p" = 6487580  Dia tri cua con tro "p" = 6487568  Gia tri cua bien ma con tro "p" dang tro toi = 10  -------------------  Gia tri cua "value" = 10  Gia tri cua "value" = 100  Gia tri cua bien ma con tro "p" dang tro toi = 100  -------------------  Gia tri cua "value" = 1000  Gia tri cua bien ma con tro "p" dang tro toi = 1000 |

**Nhận xét:**

|  |
| --- |
| * Địa chỉ của biến value chính là giá trị của con trỏ p, đều là 6487580. Lưu ý mỗi lần chạy thì giá trị địa chỉ này có thể khác nhau. * Con trỏ có thể lấy giá trị của biến mà nó đang trỏ tới bằng toán tử \*:   printf("\nGia tri cua bien ma con tro \"p\" dang tro toi = %i", \*p);   * Con trỏ có thể thay đổi giá trị của biến mà nó đang trỏ tới. Do nó mang địa chỉ của biến, khi đó nó hoàn toàn có quyền thay đổi giá trị của biến đó. Như ở ví dụ trên ta thay đổi giá trị từ 10 lên 100. |

### 2. Các lỗi thường gặp khi làm việc với con trỏ

Giả sử bạn muốn khởi tạo giá trị của con trỏ p trỏ tới địa chỉ của biến value, khi đó:

|  |
| --- |
| int value, \*p;    **/\* Sai.**   * **p cần 1 địa chỉ.** * **value không phải là cái địa chỉ. \*/**   p = value;    **/\* Sai.**   * **\*p là giá trị của biến mà con trỏ đang trỏ tới.** * **// &value là địa chỉ. \*/**   \*p = &value;    **/\* Đúng.**   * **p cần 1 địa chỉ.** * **// &value là địa chỉ của biến value.**   p = &value;    **/\* Đúng.**   * **\*p là giá trị của biến mà con trỏ đang trỏ tới.** * **value cũng là giá trị (không phải địa chỉ). \*/**   \*p = value; |

Chú ý về dấu \* ở phần khai báo và khi lấy giá trị của biến mà con trỏ đang trỏ tới:

|  |
| --- |
| #include <stdio.h>  int main() {      int c = 5;  **// Dấu \* ở đây để chúng ta biết chúng ta đang khai báo con trỏ.**  **// Không phải lấy giá trị của nó nhé**      int \*p = &c;      // Khai báo trên tương đương      // int \*p;      // p = &c;      // Nếu bạn muốn phân biệt 2 thằng này, khi khai báo có thể viết như sau:      // int\* p = &c;        // Lấy giá chỉ của biến mà con trỏ đang trỏ tới, chính là giá trị của c      printf("%i", \*p); // 5  } |

## CÁC LOẠI CON TRỎ TRONG C

Con trỏ có nhiều khái niệm nhưng dễ và chúng rất quan trọng đối với lập trình C. Các khái niệm con trỏ quan trọng sau đây nên rõ ràng đối với bất kỳ lập trình viên C nào:

|  |  |
| --- | --- |
| STT | Khái niệm & Mô tả |
| 1 | ***Con trỏ số học***  Có bốn toán tử số học có thể được sử dụng trong con trỏ: ++, -, +, - |
| 2 | ***Mảng con trỏ***  Bạn có thể xác định mảng để chứa một số con trỏ. |
| 3 | ***Con trỏ tới con trỏ***  C cho phép bạn có con trỏ trên một con trỏ, v.v. |
| 4 | ***Chuyển con trỏ đến các hàm trong C***  Truyền một đối số theo tham chiếu hoặc theo địa chỉ cho phép đối số đã truyền được thay đổi trong hàm gọi bởi hàm được gọi. |
| 5 | ***Con trỏ trả về từ các hàm trong C***  C cho phép một hàm trả về một con trỏ tới biến cục bộ, biến tĩnh và cả bộ nhớ được cấp phát động. |

### Con trỏ số học

Một con trỏ trong C là một địa chỉ, là một giá trị số. Do đó, bạn có thể thực hiện các phép toán số học trên một con trỏ giống như bạn có thể thực hiện trên một giá trị số. Có bốn toán tử số học có thể được sử dụng trên con trỏ: ++, --, +, –. Tuy nhiên khi sử dụng toán tử tăng/giảm trên biến con trỏ, nó sẽ nhảy sang phần tử liền kề chứ không phải tăng/giảm giá trị lên 1 đơn vị.

#### Tăng một con trỏ

**VD:**Tăng biến con trỏ để truy cập từng phần tử kế tiếp của mảng.

|  |
| --- |
| #include <stdio.h>  const int MAX\_SIZE = 3;  int main() {      int var[] = {10, 100, 200};      int i, \*p;      p = var; //hoặc p = &var[0];      for (i = 0; i < MAX\_SIZE; i++) {          printf("Dia chi cua var[%i] = %d\n", i, p);          printf("Gia tri cua var[%i] = %i\n", i, \*p);          p++;      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| Dia chi cua var[0] = 6422284  Gia tri cua var[0] = 10  Dia chi cua var[1] = 6422288  Gia tri cua var[1] = 100  Dia chi cua var[2] = 6422292  Gia tri cua var[2] = 200 |

#### Giảm một con trỏ

**VD:**Giảm biến con trỏ để truy cập từng phần tử kế tiếp của mảng.

|  |
| --- |
| #include <stdio.h>  const int MAX\_SIZE = 3;  int main() {        int var[] = {10, 100, 200};      int i, \*p;      p = &var[2]; //hoặc p = &var[0];      for (i = MAX\_SIZE-1; i >= 0; i--) {          printf("Dia chi cua var[%i] = %d ", i, p);          printf("Gia tri cua var[%i] = %i\n", i, \*p);          p--;      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| Dia chi cua var[2] = 6422292 Gia tri cua var[2] = 200  Dia chi cua var[1] = 6422288 Gia tri cua var[1] = 100  Dia chi cua var[0] = 6422284 Gia tri cua var[0] = 10 |

#### So sánh con trỏ

Và chúng ta cũng có một vài tính chất sau của phép so sánh. Lưu ý là các bạn không phải học thuộc lòng, mà phải đọc hiểu từng phần.

* Phép so sánh ngang bằng dùng để kiểm tra 2 con trỏ có trỏ vào cùng 1 vùng nhớ hay không, hoặc kiểm tra 1 con trỏ có phải là đang trỏ vào NULL hay không (trong trường hợp cấp phát động, mở file, mở resource, vv).
* Phép so sánh lớn hơn nhỏ hơn: >, =, <= sử dụng để kiểm tra về độ thấp cao giữa 2 địa chỉ. Con trỏ nào nhỏ hơn thì trỏ vào địa chỉ thấp hơn.
* Được quyền so sánh mọi con trỏ với 0, vì 0 chính là NULL.
* Ngoài ra thì khi so sánh 2 con trỏ hoặc con trỏ với 1 địa chỉ xác định (số nguyên) cần có sự tương xứng về kiểu dữ liệu
* Con trỏ void có thể đem ra so sánh với tất cả các con trỏ khác

#### Phép gán con trỏ

Một số điều sau đây các bạn phải ghi nhớ.

* Tất cả con trỏ đều có phép gán.
* Như phần khởi tạo, phép gán yêu cầu vế trái là một con trỏ, và vế phải là một địa chỉ.
* Phép gán yêu cầu sự tương xứng về kiểu dữ liệu, nếu không chúng ta phải ép kiểu, các bạn biết ép kiểu không?
* Phép gán với con trỏ kiểu void không cần phải tương xứng kiểu dữ liệu.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {  int a = 100;  int \*p;  p = &a;  printf("\*p: %d\n", \*p); // kết quả của \*p là 100 vì p trỏ tới a      return 0;  } |

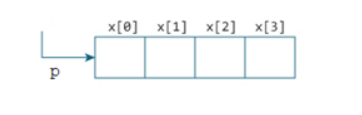
**Kết quả**

|  |
| --- |
| \*p: 100 |

### Mảng con trỏ

**Mảng con trỏ** là một mảng chứa các phần tử là con trỏ. Ta khai báo 1 con trỏ và 1 mảng 4 phần tử như sau:

|  |
| --- |
| int x[4];  int \*p;  p = x;     // gán con trỏ p trỏ đến địa chỉ của phần tử đầu tiên của mảng             // hoặc có thể gán như sau p = &x[0], 2 cách gán này như nhau |

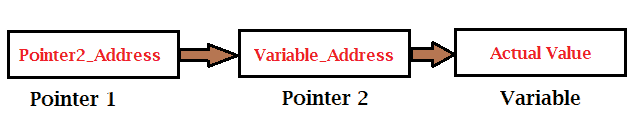


Với mảng trong ảnh phía trên, ta có:

* + &x[0] và p có cùng giá trị, và x[0] hay \*p hay p[0] là tương đương nhau.
  + &x[1] tương đương với p+1 và x[1] tương đương với \*(p+1) và p[1].
  + &x[2] tương đương với p+2 và x[2] tương đương với \*(p+2) và p[2].

Tóm lại, **&x[i] tương đương với p+**i và **x[i] tương đương với \*(p+i) và p[i].**

### Con trỏ trỏ tới một con trỏ (pointer to pointer)



Theo hình trên, con trỏ đầu tiên sẽ chứa địa chỉ của con trỏ thứ 2, và con trỏ thứ 2 lại chứa địa chỉ của biến Variable.

A picture containing text, diagram, screenshot, line

Description automatically generated

Theo hình trên, con trỏ ptr2 sẽ chứa địa chỉ của con trỏ ptr1, và con trỏ ptr1 sẽ chứa địa chỉ của biến num. Ta gọi ptr2 là con trỏ trỏ tới con trỏ (pointer to pointer). Đây là cách hiểu đơn giản của pointer to pointer.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main () {      int var;      int \*ptr;      int \*\*pptr;      var = 3000;      /\* ptr lấy địa chỉ của biến var \*/      ptr = &var;      /\* pptr lấy địa chỉ của biến ptr \*/      pptr = &ptr;      /\* in giá trị sử dụng con trỏ pptr \*/      printf("Gia tri cua var = %d\n", var );      printf("Gia tri cua \*ptr = %d\n", \*ptr );      printf("Gia tri cua \*\*pptr = %d\n", \*\*pptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **Gia tri cua var = 3000**  **Gia tri cua \*ptr = 3000**  **Gia tri cua \*\*pptr = 3000** |

### Chuyển con trỏ đến các hàm trong C

#### Truyền con trỏ biến vào hàm trong C

Ở bài trước chúng ta có tìm hiểu về cách truyền tham chiếu và tham trị vào 1 hàm.

Và khi chúng ta truyền vào tham trị (truyền vào giá trị) chứ không phải truyền vào tham chiếu (địa chỉ) thì mặc dù trong hàm đã thay đổi giá trị các biến nhưng sau khi hàm thực hiện xong thì các giá trị vẫn chưa thể thay đổi được. Và chúng ta sẽ phải sửa lại bằng cách **truyền vào tham chiếu** chính là **các con trỏ.**

**VD:**

|  |
| --- |
| #include <stdio.h>    void hoanVi\_thamchieu(int \**a*, int \**b*)  {      int temp = \**a*;      \**a* = \**b*;      \**b* = temp;  }    void hoanVi\_thamtri(int *a*, int *b*)  {      int temp = *a*;  *a* = *b*;  *b* = temp;  }    int main()  {      int a = 10, b = 1000;        printf("Truoc khi goi ham hoan vi: a = %d, b = %d \n", a, b);        hoanVi\_thamtri(a, b);      printf("Sau khi goi ham hoan vi: a = %d, b = %d \n", a, b);        hoanVi\_thamchieu(&a, &b);      printf("Sau khi goi ham hoan vi: a = %d, b = %d \n", a, b);        return 0;  } |

**Kết quả**

|  |
| --- |
| **Truoc khi goi ham hoan vi: a = 10, b = 1000**  **Sau khi goi ham hoan vi: a = 10, b = 1000**  **Sau khi goi ham hoan vi: a = 1000, b = 10** |

Ở đây chúng ta thấy rằng ở hàm **void hoanVi\_thamchieu(int \*a, int \*b)** khai báo là 2 con trỏ dùng đến trỏ tới địa chỉ 2 biến a,b trong bộ nhớ RAM. Do đó chúng ta cũng cần truyền vào 2 địa chỉ của biến a và b trên RAM. Khi hàm khởi chạy nó sẽ tác động trực tiếp đến giá trị của các vùng địa chỉ này (tác động trực tiếp lên a và b) mà không phải là 1 bản sao nào cả. Do đó sau khi kết thúc hàm, thì giá trị của 2 biến a và b cũng được thay đổi theo.

#### Truyền con trỏ mảng vào hàm trong C

Ở bài mảng chúng ta đã biết rằng để truyền 1 mảng vào hàm chúng ta có thể thông qua 1 con trỏ. Trong mục này chúng ta sẽ cùng đi sâu thêm để vận dụng nhé.

**VD: Viết hàm nhập mảng và in mảng**

|  |
| --- |
| #include <stdio.h>  #include <string.h>  int mang[100] = {0};  void NhapMang(int \**pt*, int \**n*)  {      printf("Nhap vao so phan tu mang: ");      scanf("%d", *n*);      printf("Nhap Mang\n");      for (int i = 0; i < \**n*; i++)      {          printf("mang[%d]= ", i);          scanf("%d", *pt*);   **// gán giá trị cho phần tử mảng thông qua con trỏ**  *pt* = *pt*+1;         **// trỏ con trỏ pt đến phần tử mảng tiếp theo**      }  }  void InMang(int \**pt*, int *n*)  {      printf("In Mang\n");      for (int i = 0; i < *n*; i++)      {          printf("mang[%d] = %d\n", i,\**pt*); **// có thể thay \*pt bằng pt[i]**  *pt* = *pt* + 1;      }  }  int main()  {      int phan\_tu\_mang;      NhapMang(mang,&phan\_tu\_mang);      InMang(mang, phan\_tu\_mang);  } |

**Kết quả:**

|  |
| --- |
| Nhap vao so phan tu mang: 5  Nhap Mang  mang[0]= 1  mang[1]= 2  mang[2]= 3  mang[3]= 4  mang[4]= 5  In Mang  mang[0] = 1  mang[1] = 2  mang[2] = 3  mang[3] = 4  mang[4] = 5 |

#### Truyền con trỏ chuỗi vào hàm trong C

### Con trỏ trả về từ các hàm trong C

Bình thường khi bạn viết một hàm, bạn phải có kiểu trả về cho hàm đó, điều này quá đỗi bình thường trong C. Và trong phần này, bạn sẽ học một kiểu trả về mới của một hàm, đó chính là pointer hoặc nói cách khác là địa chỉ. Đúng vậy, một hàm có thể trả về một pointer(địa chỉ).

**Cú pháp:**

|  |
| --- |
| Kieu tra ve \* Ten Ham() {     // Các câu lệnh  } |

**VD: Viết chương trình tìm và in ra số lớn nhất trong 2 số**

|  |
| --- |
| #include <stdio.h>  int \* timSoLonNhat(int *x*, int *y*)  {      if(*x*>*y*)      {          return &*x*;     // trả về 1 địa chỉ      }      else      {          return &*y*; // trả về 1 địa chỉ      }  }  int main()  {      int \*result;      int a = 11;      int b = 15;      result = timSoLonNhat(a,b); // truyền vào giá trị của a và b      printf("So lon nhat = %d\r\n",\*result);      return 0;  } |

**Kết quả**

|  |
| --- |
| **Segmentation fault (core dumped)** |

**Vì sao là như vậy ?**

Các bạn để ý rằng trong hàm timSoLonNhat có 2 biến x và y được khởi tạo để chứa giá trị được truyền vào (giá trị của a và b). Và vì được khai báo trong hàm nên x và y là 2 biến cục bộ (sẽ bị giải phóng khi ra khỏi hàm). Do đó khi hàm trả về địa chỉ của 1 trong 2 biến x hoặc y thì nó chỉ có ý nghĩa trong hàm timSoLonNhat mà thôi. Ra khỏi hàm x và y sẽ biến mất do đó địa chỉ của chúng cũng không có ý nghĩa. Và con trỏ result vô tình gán đến 1 vùng địa chỉ không có ý nghĩa nên gây ra lỗi khi ta truy cập vào vùng địa chỉ đó.

**Vậy nếu đúng phải như thế nào ?**

|  |
| --- |
| #include <stdio.h>  int\* timSoLonNhat(int \**x*, int \**y*)  {      if(\**x*>\**y*)      {          return *x*; // trả về địa chỉ con trỏ x trỏ tới      }      else      {          return *y*; // trả về địa chỉ con trỏ y trỏ tới      }  }  int main()  {      int \*result;      int a = 11;      int b = 15;      result = timSoLonNhat(&a,&b); // truyền vào địa chỉ của a và b      printf("So lon nhat = %d\r\n",\*result);      return 0;  } |

**Kết quả**

|  |
| --- |
| **So lon nhat = 15** |

Ví dụ trên là một hàm trả về một con trỏ. Điều này hoàn toàn không có gì phức tạp, con trỏ thì chứa địa chỉ, cho nên return một con trỏ đồng nghĩa với return một địa chỉ. Bạn hãy để ý là biến x, y (biến local trong hàm) sẽ bị phá hủy (destroy) ngay khi ra khỏi hàm, cho nên khi con trỏ result “lấy” địa chỉ trả về của hàm timSoLonNhat, địa chỉ này đã bị phá hủy rồi, nên chương trình sẽ gây ra lỗi vì con trỏ không biết đang trỏ đến đâu, có thể là một địa chỉ rác. Cho nên khi chúng ta sử dụng hàm để trả về con trỏ thì phải đặc biệt lưu ý điều này.

Thông thường chúng ta sử dụng hàm để trả về con trỏ trong 2 trường hợp là **biến static** hoặc **cấp phát động**.

**VD về sử dụng biến static**

|  |
| --- |
| #include <stdio.h>  int \* timSoLonNhat(int *x*, int *y*)  {      static int max;      if(*x*>*y*)      {          max = *x*;      }      else      {          max = *y*;      }      return &max;  }  int main()  {      int \*result;      int a = 11;      int b = 15;      result = timSoLonNhat(a,b); // truyền vào giá trị của a và b      printf("So lon nhat = %d\r\n",\*result);      return 0;  } |

**Kết quả:**

|  |
| --- |
| **So lon nhat = 15** |

**VD về sử dụng cấp phát động**

|  |
| --- |
| #include <stdio.h>  int \* timSoLonNhat(int *x*, int *y*)  {      int \*max = (int \*) malloc(sizeof( int ));      if(*x*>*y*)      {          \*max = *x*;      }      else      {          \*max = *y;*      }      return max;  }  int main()  {      int \*result;      int a = 11;      int b = 15;      result = timSoLonNhat(a,b); // truyền vào giá trị của a và b      printf("So lon nhat = %d\r\n",\*result);      return 0;  } |

**Kết quả**

|  |
| --- |
| **So lon nhat = 15** |