Bài 2: Pointer

Con trỏ

Đã đến lúc chúng ta tìm hiểu về con trỏ. Hãy ra hít một hơi thật sâu trước khi bắt đầu vì tôi biết bài học này chắc chắn sẽ không khiến bạn thấy thú vị. Nhưng con trỏ là một khái niệm được sử dụng rất thường xuyên trong C. Nói về tầm quan trọng, chúng ta không thể nào lập trình trên ngôn ngữ C mà không dùng đến con trỏ, và bạn cũng đã từng dùng nó mà không biết.

Phần lớn những người bắt đầu học C thường xuyên vấp ngã trong phần kiến thức về con trỏ. Và tôi hi vọng bài học này sẽ giúp các bạn không nằm trong số đó. Hãy tập trung gấp đôi bình thường và bỏ thêm thời gian để hiểu rõ từng biểu đồ, ví dụ có trong bài học này.

Một vấn đề nan giải

Đây là một trong những vấn đề lớn liên quan đến con trỏ, các bạn mới bắt đầu thường bị nhầm lẫn, cảm thấy khó khăn trong việc nắm vững cách hoạt động và sử dụng.

"Con trỏ rất cần thiết, và chúng ta sẽ thường xuyên dùng đến nó, hãy tin tôi !"

Tôi sẽ cho bạn xem một ví dụ mà các bạn không thể nào giải quyết được nếu không sử dụng đến con trỏ. Đây cũng là tiêu điểm của bài học này, tôi sẽ hướng dẫn cách giải quyết ở cuối bài học.

Đây là vấn đề: Tôi muốn viết một function trả về hai giá trị. Việc này là **không thể** vì mỗi function chỉ có thể trả về duy nhất một giá trị.

C code:

```
int function ( )
{
   return giatri;
}
```

Nếu ta khai báo function với type *int*, thì ta sẽ nhận được một số dạng *int* (nhờ vào instruction *return*).
Chúng ta cũng đã học cách viết một function không trả về bất cứ giá trị nào với từ khóa *void*:

C code:

```
void function( )
{

}
```

Nhưng để nhận được hai giá trị trả về cùng lúc thật sự là việc không thể. Chúng ta không thể sử dụng hai *return* cùng lúc.

Giả sử tôi muốn viết một function, trong parameter tôi sẽ cho nó một giá trị tính bằng phút, tôi muốn nó chuyển thành giờ và phút tương ứng:

1. Nếu ta đưa vào giá trị 45, function sẽ trả về 0 giờ và 45 phút.
2. Nếu ta đưa vào giá trị 60, function sẽ trả về 1 giờ và 0 phút.
3. Nếu ta đưa vào giá trị 60, function sẽ trả về 1 giờ và 30 phút .

Nhìn có vẻ khá đơn giản, nhưng hãy cùng test đoạn code sau:

C code:

```
#include <stdio.h>
#include <stdlib.h>
        /* Toi dat cac prototypes tren cung. Vi day la mot chuong trình kha nho nen toi khong
        dat chung trong mot file.h, nhung trong mot chuong trinh that su, toi se dat chung trong
        mot file.h nhu da huong dan o bai truoc*/
void chuyenDoi(int gio, int phut);
int main (int argc, char *argv[ ])
{
   int gio = 0, phut = 90;
            /*Chung ta co bien so "phut" giá trị 90. Sau khi ket thuc function, toi muon bien so
        "gio" nhan gia tri 1 và bien so "phut" nhan gia tri 30 */
   chuyenDoi(gio, phut);
   printf ("%d gio va %d phut", gio, phut);
   return 0;
}
void chuyenDoi(int gio, int phut)
{
   gio= phut/ 60;  // 90 / 60 = 1
   phut= phut% 60; // 90 % 60 = 30
}
```

Và đây là kết quả:

Console

```
0 gio va 90 phut
```

Ặc... chương trình đã không hoạt động. Vì sao vậy?

Khi bạn gửi giá trị của một biến số vào vị trí parameter của một function, một bản sao của biến số này được tạo ra. Nói cách khác, biến số "*gio*" trong function *chuyenDoi* không phải là biến số "*gio*" trong function *main*! Nó chỉ là bản sao!

Function *chuyenDoi* đã thực hiện nhiệm vụ của nó. Trong function *chuyenDoi*, những biến số "*gio*" và "*phut*" nhận giá trị chính xác: 1 và 30.

Nhưng sau đó, function kết thúc khi dấu ngoặc } đóng lại. Như ta đã học ở bài học trước, tất cả những biến số tạo ra trong một function sẽ bị xóa đi khi function đó kết thúc. Và ở đây, biến số *gio* và *phut* đã bị xóa đi. Sau đó chương trình tiếp tục phần tiếp theo của *main*, và ở đó biến số *gio* và *phut* của *main* giá trị vẫn là 0 và 90. Đó là lí do bạn thất bại!

Cần ghi thêm ở đây, function tạo ra một bản sao cho biến số ta gửi vào nó, nên bạn không cần phải gọi tên biến số đó chính xác giống như cách bạn gọi ở main.

Để rõ ràng hơn, bạn xem đoạn code sau:

```
void chuyenDoi (int g, int p)
```

(g thay cho gio và p thay cho phút)

Và tiếp theo, hãy thử tìm nhiều cách khác sửa đổi chương trình trên, như trả về một giá trị sau khi kết thúc function (sử dụng *return* và thay đổi *type* function thành *int*), bạn chỉ nhận được một trong hai giá trị bạn cần. Bạn không thể nào nhận được cùng lúc hai giá trị. Và bạn tuyệt đối không được sử dụng biến số *global,* lí do tôi đã giải thích ở bài trước.

Và đó là vấn đề khó khăn đặt ra, vậy con trỏ sẽ giải quyết vấn đề trên như thế nào?

Địa chỉ trong bộ nhớ
Nhắc lại kiến thức

Bạn còn nhớ bài học về những biến số không?
Dù có hay không, tôi vẫn khuyến khích bạn xem lại phần đầu của bài học, phần "Công việc của bộ nhớ".

Trong đó có một biểu đồ khá quan trọng mà tôi cần nhắc lại trước khi dạy bạn những kiến thức mới:

Cách sắp xếp trong bộ nhớ (RAM)

Đó là cách trình bày một RAM trong máy tính của bạn.

Hãy đọc kĩ từng dòng trong biểu đồ. Dòng đầu tiên tương ứng với "ô" đầu tiên của bộ nhớ (RAM). Mỗi ô tương ứng với một số, là địa chỉ của nó (address)! Bộ nhớ chứa một số lượng lớn địa chỉ, bắt đầu từ địa chỉ 0 đến một số nào đó (một số vô cùng lớn, số lượng địa chỉ phụ thuộc vào dung lượng bộ nhớ được lắp đặt trong từng máy tính).

Mỗi địa chỉ có thể chứa một số. Một và chỉ một. Ta không thể nào chứa 2 số trong cùng một địa chỉ.

Bộ nhớ của bạn tạo ra chỉ để chứa những con số. Nó không thể chứa chữ cái cũng như đoạn văn. Để giải quyết vấn đề này, người ta tạo ra những bảng mã chứa trong đó số và chữ cái tương ứng.

Ví dụ "Số 89 tương ứng với chữ cái Y". Vấn đề này sẽ được giải thích rõ hơn ở bài học sau. Bây giờ, chúng ta chỉ tập trung vào cách hoạt động của bộ nhớ.

Địa chỉ và giá trị

Khi bạn tạo ra một biến số *tuoi* type *int*, lấy ví dụ:

C code:
```
int tuoi = 10;
```

... chương trình của bạn sẽ yêu cầu hệ điều hành (ví dụ là Windows) quyền sử dụng một ít bộ nhớ. Hệ điều hành sẽ trả lời bằng cách đưa ra địa chỉ bộ nhớ được phép chứa con số bạn cần.
Đây cũng là một trong những nhiệm vụ chính của hệ điều hành:

Khi chúng ta yêu cầu mượn bộ nhớ cho chương trình. Máy tính giống như ông chủ, nó điều hành từng chương trình và kiểm tra xem chúng có quyền sử dụng bộ nhớ tại vị trí được cấp hay không.

Và đây là một trong những nguyên nhân khiến máy tính bạn bị đơ: Nếu chương trình đột nhiên hoạt động trên một vùng bộ nhớ không cho phép. Hệ điều hành (OS) sẽ từ chối và dừng ngay chương trình, giống như nói với bạn "Mày nghĩ ai là chủ ở đây?" Người dùng, sẽ nhìn thấy một cửa sổ hiện lên thông báo dạng "Chương trình bị dừng lại do thực hiện một công việc không được phép".

Quay trở lại với biến số *tuoi*. Giá trị 10 được đưa vào một vị trí nào đó trong bộ nhớ, lấy ví dụ nó được đưa vào địa chỉ 4655. Và điều xảy ra ở đây là (nhiệm vụ của compiler), từ *tuoi* trong chương trình sẽ thay thế bằng địa chỉ 4655 khi được chạy.

Việc đó giống như, mỗi khi bạn điền vào *tuoi* trong code source, chúng sẽ được chuyển thành 4655, và máy tính sẽ biết được cần đến địa chỉ nào trong bộ nhớ để lấy giá trị . Và ngay sau đó, máy tính xem giá trị được chứa trong địa chỉ 4655 và trả lời chúng ta "biến số tuoi co giá trị là 10"!

Và để lấy giá trị một biến số, đơn giản chỉ cần đánh tên của biến số đó vào code source. Nếu ta muốn hiển thị tuổi, ta có thể sử dụng function *printf*:

C code:
```
printf ("Bien so tuoi co gia tri la : %d", tuoi);
```

Không có điều gì mới với dòng code trên đúng ko.

Khuyến mãi thêm!

Bạn đã biết cách hiển thị giá trị của một biến số, nhưng bạn có biết chúng ta cũng có thể hiển thị địa chỉ của biến số đó?

...Đương nhiên là bạn chưa biết rồi

Để hiển thị địa chỉ của một biến số, chúng ta cần sử dụng kí hiệu *%p* (p ở đây viết tắt của từ pointer) trong *printf*. Mặt khác, chúng ta phải đưa vào *printf* địa chỉ của biến số đó và để làm việc này, bạn cần phải đặt kí hiệu *&* trước biến số đó (*tuoi*), giống như cách tôi hướng dẫn bạn sử dụng *scanf*, xem code sau:

C code:

```
printf ("Dia chi cua bien so tuoi la %p", &tuoi);
```

Kết quả

Console

```
Dia chi cua bien so tuoi la 0023FF74
```

Đó là địa chỉ của biến số *tuoi* trong thời điểm chương trình hoạt động. Vâng, 0023FF74 là một số, nó đơn giản chỉ được viết trên hệ hexadecimal (thập lục phân), thay vì hệ decimal (thập phân) mà chúng ta thường sử dụng. Nếu bạn thay kí hiệu *%p* thành *%d*, bạn sẽ nhận được một số thập phân mà bạn biết.

Nếu bạn chạy chương trình này trên máy tính của bạn, địa chỉ sẽ khác hoàn toàn.Tất cả phụ thuộc vào phần trống có trong bộ nhớ, chương trình bạn đang dùng,... Hoàn toàn không có khả năng báo trước địa chỉ nào của biến số sẽ được cấp. Nếu bạn thử chạy chương trình liên tục nhiều lần, địa chỉ có thể sẽ không đổi trong thời điểm đó. Nhưng nếu bạn khởi động lại máy tính, chương trình chắc chắn sẽ hiển thị một giá trị khác.

Vậy chúng ta sẽ làm gì với tất cả những thứ đó?

Tôi cần bạn nắm vững những điều sau:

- *tuoi*: tượng trưng cho **giá trị** của biến số.
- *&tuoi*: tượng trưng cho **địa chỉ** của biến số.

Với *tuoi*, máy tính sẽ đọc và gửi lại giá trị của biến số.
Với *&tuoi*, máy tính sẽ nói với chúng ta ở địa chỉ nào sẽ tìm thấy biến số.

Cách sử dụng pointers (con trỏ)

Đến bây giờ, bạn chỉ có thể tạo biến số để chứa các số hạng. Và sau đây chúng ta sẽ học cách tạo ra những biến số chứa địa chỉ của chúng, những biến số này gọi là con trỏ.

Nhưng... Địa chỉ cũng là một số đúng không? Như vậy số này cần phải chứa trong một biến số khác và cứ như thế nó sẽ lặp lại mãi sao?

Chính xác. Nhưng các con số này sẽ có một kí hiệu đặc biệt để nhận biết: địa chỉ của một biến số khác trong bộ nhớ.

Cách tạo một con trỏ

Để tạo một biến số dạng con trỏ, ta cần phải thêm kí tự * trước tên của biến số.

C code:

```
int *pointer;
```

Bạn cần biết rằng chúng ta cũng có thể viết:

```
int* pointer;
```

vẫn hoạt động tương tự như trên. Nhưng cách viết đầu được khuyến khích hơn. Bởi vì trong trường hợp bạn cần khai báo cùng lúc nhiều con trỏ trong cùng một dòng, bạn bắt buộc phải đặt * trước mỗi tên con trỏ:

```
int *pointer1, *pointer2, *pointer3;
```

Giống như điều tôi dạy bạn khi khai báo biến số, bạn cần cho nó giá trị ngay khi khởi tạo, rất quan trọng, bằng cách cho nó giá trị 0 (lấy ví dụ với biến số). Và đối với con trỏ, điều này còn quan trọng hơn nữa! Để khởi tạo con trỏ, có nghĩa là cho nó một giá trị mặc định, người ta không dùng giá trị 0 mà dùng từ khóa *NULL* (phải được viết hoa):

C code:

```
int *pointer = NULL;
```

Bạn đã khởi tạo một con trỏ giá trị *NULL*. Như vậy, bạn chắc rằng con trỏ của bạn không chứa địa chỉ nào.

Việc này diễn ra như thế nào? Đoạn mã trên sẽ đặt trước một chỗ trong bộ nhớ, giống như cách bạn tạo ra một biến số thông thường. Nhưng điều thay đổi ở đây là giá trị của con trỏ chỉ dùng để chứa địa chỉ của một biến số khác.

Vậy thử xem vơi địa chỉ của *tuoi* thì sao?

Và đây là cách chỉ ra địa chỉ của một biến số (*tuoi*) dựa trên giá trị của nó (bằng cách sử dụng kí tự &), nào bắt đầu thôi!

C code:

```
int tuoi = 10;
int *pointerTuoi = &tuoi;
```

Dòng thứ nhất : "Tạo một biến số type int có giá trị là 10".
Dòng thứ hai : "Tạo một con trỏ có giá trị là địa chỉ của biến số tuoi".

Dòng thứ hai thực hiện cùng lúc hai việc. Nếu bạn thấy phức tạp nên không muốn gộp hai việc với nhau, tôi sẽ tách biệt chúng bằng cách chia thành hai giai đoạn, xem đoạn code sau :

C code:

```
int tuoi = 10;
int *pointerTuoi; // 1) co nghia la "Toi tao mot con tro poiterTuoi"
pointerTuoi = &tuoi; // 2) co nghia la "con tro pointerTuoi chua dia chi cua bien so tuoi"
```

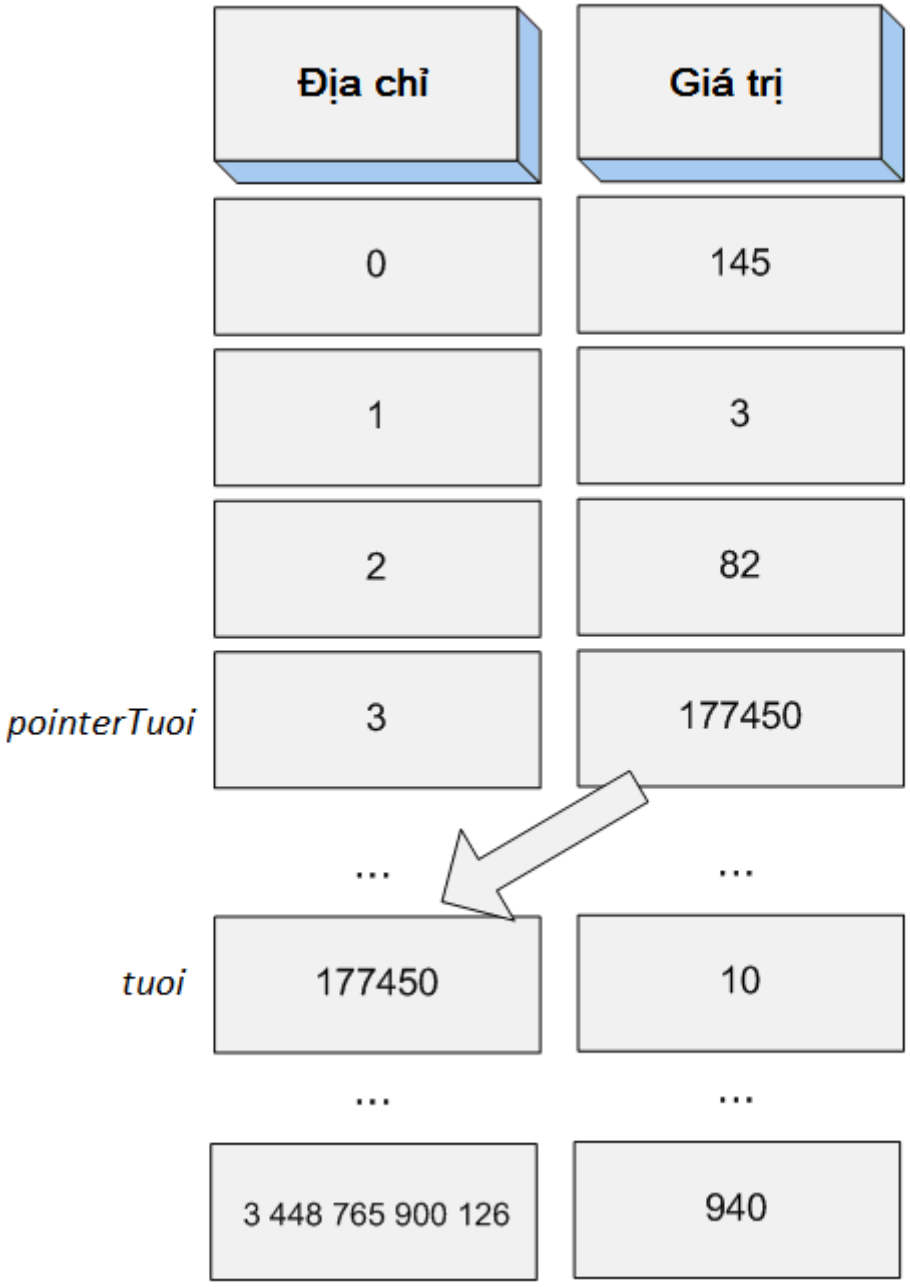
Bạn cần nhớ rằng không có type "pointer" như type int hay double. Người ta không ghi như sau: pointer pointerTuoi;

Thay vì vậy, chúng ta sẽ sử dụng kí tự *, sau *int*. Tại sao lại như vậy? Thật ra, chúng ta cần phải chỉ rõ type của biến số mà con trỏ sẽ chứa địa chỉ của nó.

Ở trên, *pointerTuoi* sẽ chứa địa chỉ của biến số *tuoi* (type là *int*), vậy con trỏ phải có type *int** Nếu biến số *tuoi* có type là *double*, ta phải viết *double *pointerTuoi*.

Giá trị của con trỏ *pointerTuoi* chỉ ra địa chỉ của biến số *tuoi*.

Hình sau tóm tắt lại những gì đã diễn ra trong bộ nhớ:



Trong biểu đồ trên, biến số *tuoi* được đặt vào ô địa chỉ 177450 (và bạn thấy tại đó giá trị tương ứng là 10), và con trỏ *pointerTuoi* được đặt vào ô địa chỉ 3 (tất cả địa chỉ đều được chọn ngẫu nhiên, và các địa chỉ trong biểu đồ cũng do tôi tự viết ra😋).

Khi con trỏ được tạo ra, hệ điều hành sẽ dành một ô trong bộ nhớ giống như cách nó tạo ra với biến số *tuoi*. Khác nhau là giá trị của *pointerTuoi* là địa chỉ của biến số *tuoi*.

Chúng ta bắt đầu tiến vào thế giới huyền diệu của những con trỏ. Thế giới bí mật của những chương trình viết trên ngôn ngữ C (C++)


Ok, nhưng... nó dùng để làm gì ?

Hiển nhiên, nó không giúp máy tính bạn biến đổi thành máy làm ra café. Chỉ là, ta có một con trỏ *pointerTuoi* chứa địa chỉ của biến số *tuoi*. Hãy dùng *printf* xem thử nó chứa gì trong đó:

C code:

```
int tuoi = 10;
int *pointerTuoi = &tuoi;
printf ("%d", pointerTuoi);
```

Console

```
177450
```

uhm, thật sự điều này không có gì ngạc nhiên lắm. Người ta yêu cầu giá trị chứa trong *pointerTuoi* và đó là địa chỉ của biến số *tuoi* (177450).
Vậy làm sao có được giá trị của biến số mà *pointerTuoi* chỉ vào?
Chúng ta phải đặt kí tự * trước tên của con trỏ:

C code:

```
int tuoi = 10;
int *pointerTuoi = &tuoi;

printf ("%d", *pointerTuoi);
```

Console

```
10
```

Đó, bạn làm được rồi đấy! Bằng cách đặt kí tự * trước tên con trỏ, ta nhận được giá trị của biến số *tuoi*.

Nếu chúng ta sử dụng kí tự & trước tên của con trỏ, chúng ta sẽ nhận được địa chỉ để tìm thấy con trỏ (trong trường hợp này là 3).

Vậy tôi đạt được điều gì ở đây? Nãy giờ tôi chỉ thấy các vấn đề càng lúc càng rắc rối thêm. Tôi thấy không cần thiết hiển thị giá trị của biến số *tuoi* bằng con trỏ! Trước đây chúng ta vẫn có thể hiển thị giá trị của biến số mà đâu cần đến con trỏ.

Đây là một câu hỏi chính đáng (mà bạn bắt buộc phải đặt ra cho bản thân). Sau tất cả những điều rắc rối bạn vừa được học, hiển nhiên là bạn muốn biết tác dụng của nó, nhưng tại thời điểm này, tôi khó có thể giải thích hết, qua các bài học sau, từng chút một, các bạn sẽ thấy nó không đơn giản được tạo ra chỉ để làm mọi thứ phức tạp thêm.

Xin bạn hãy bỏ qua một bên cái cảm giác khó chịu tôi tạo ra cho bạn ("Tất cả mọi thứ ở trên chỉ để làm những việc này thôi sao?").

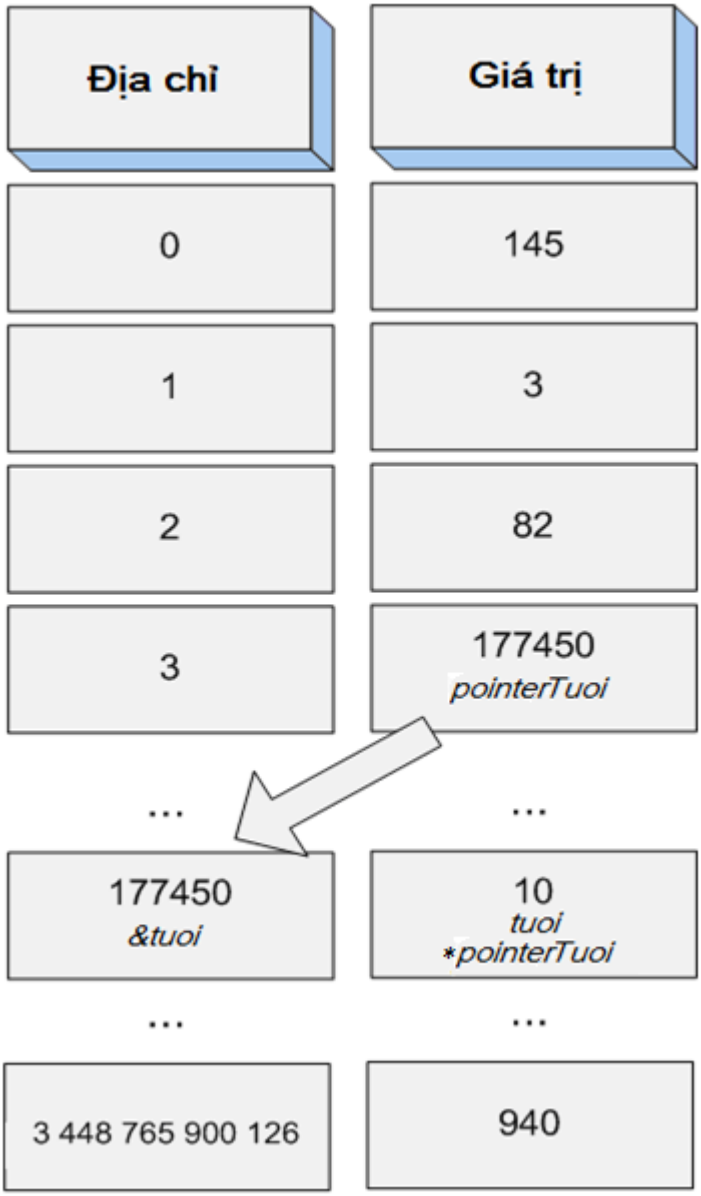
Nếu bạn hiểu được nguyên lý hoạt động, thì chắc chắn, những thắc mắc sẽ được sáng tỏ trong các bài học sau.

Những điều cần nắm vững

Đây là những điều mà bạn cần hiểu và nắm vững trước khi tiếp tục bài học:

- Đối với một biến số, lấy ví dụ biến số *tuoi*:
    - *tuoi* có nghĩa là: "Tôi muốn giá trị của biến số *tuoi*",
    - *&tuoi* có nghĩa là: "Tôi muốn địa chỉ để tìm thấy biến số *tuoi*";
- Đối với một con trỏ, lấy ví dụ con trỏ *pointerTuoi*:
    - *pointerTuoi* có nghĩa là: "Tôi muốn giá trị của con trỏ *pointerTuoi*" (giá trị này là địa chỉ của một biến),
    - **pointerTuoi* có nghĩa là: "Tôi muốn giá trị của biến số mà con trỏ *pointerTuoi* chỉ vào".

Để có thể hiểu được 4 điểm chính trên. Bạn cần test nhiều lần để hiểu cách nó hoạt động. Biểu đồ sau đây giúp bạn có thể hiểu rõ hơn:



Chú ý không nhầm lẫn các ý nghĩa của kí tự * Khi bạn khai báo một con trỏ, * có tác dụng chỉ ra bạn muốn tạo ra một con trỏ:

```
int *pointerTuoi;
```

Còn trong printf:

```
printf ("%d",*pointerTuoi);
```

điều này không phải "tôi muốn tạo một con trỏ" mà là "tôi muốn giá trị của biến số mà con trỏ chỉ vào".

Tất cả những điều trên là cơ bản. Bạn phải học thuộc lòng và tất nhiên phải hiểu rõ. Đừng ngại đọc đi đọc lại nhiều lần. Đừng xấu hổ nếu không hiểu ngay được bài học khi chỉ đọc qua lần đầu tiên, có nhiều vấn đề chúng ta cần nhiều ngày để có thể hiểu rõ và đôi khi cần nhiều tháng để có thể sử dụng thành thạo.

Nếu bạn có cảm giác không theo kịp, thì hãy nghĩ đến những bậc thầy trong việc lập trình: không ai trong số họ có thể hiểu rõ hoàn toàn hoạt động của con trỏ trong lần đầu tiên. Nếu có một người như vậy tồn tại, bạn hãy giới thiệu với tôi nhé.

Cách sử dụng con trỏ trong một function

Điều khá thú vị ở con trỏ là chúng ta có thể sử dụng chúng trong các function để có thể thay đổi trực tiếp giá trị của biến số trong bộ nhớ chứ không phải một bản sao như bạn đã thấy ở đoạn đầu bài học.

Vậy nó hoạt động như thế nào? Có rất nhiều cách thức để sử dụng. Đây là ví dụ đầu tiên:

C code:

```
void triplePointer(int *pointerSoHang);

int main (int argc, char *argv[ ])
{
   int soHang = 5;

   triplePointer(&soHang); // Ta gui dia chi cua soHang vao function
   printf ("%d", soHang); /* Ta hien thi bien so soHang. Va function da truc tiep thay doi gia tri cua bien so vi no biet dia chi cua bien so nay */
   return 0;
}

void triplePointer(int *pointerSoHang)
{
   *pointerSoHang *= 3; // Ta x3 gia tri cua so hang duoc dua vao
}
```

Console

```
15
```

Function *triplePointer* nhận vào parameter giá trị *type int* * (đó là một con trỏ chỉ vào một biến số *type int*).

Và đây là những gì diễn ra theo thứ tự, bắt đầu bởi function *main*:

1. Một biến số *soHang* được tạo ra trong *main*. Khởi tạo với giá trị 5.

2. Ta gọi function *triplePointer*. Ta gửi vào parameter địa chỉ của biến số.

3. Function *triplePointer* nhận địa chỉ là giá trị của *pointerSoHang*. Và trong funtion *triplePointer*, ta có một con trỏ *pointerSoHang* chứa địa chỉ của biến số *soHang*

4. lúc này, ta có một con trỏ chỉ lên biến số *soHang*, ta đã có thể thay đổi trực tiếp giá trị của biến số *soHang* trong bộ nhớ! Chỉ cần dùng **pointerSoHang* để điều chỉnh giá trị của biến số *soHang*! Ở ví dụ trên, người ta chỉ đơn giản thực hiện: nhân 3 lần giá trị của biến số *soHang*.

5. kết thúc bằng *return* trong function *main*, lúc này *soHang* đã có giá trị 15 vì function *triplePointer* đã trực tiếp thay đổi giá trị của nó.

Tất nhiên, tôi có thể thực hiện *return* để trả về giá trị như cách chúng ta đã học trong bài học về function. Nhưng điều thú vị ở đây là, bằng cách sử dụng con trỏ, chúng ta có thể thay đổi giá trị của nhiều biến số trong bộ nhớ (có nghĩa là "chúng ta có thể trả về nhiều giá trị"). Không còn giới hạn một giá trị duy nhất được trả về nữa !

Vậy return còn giá trị sử dụng gì khi người ta đã có thể dùng con trỏ để thay đổi giá trị ?

Điều này phụ thuộc vào bạn và chương trình bạn viết. Chúng ta cần hiểu là cách dùng *return* để trả về giá trị là một cách viết khá đẹp và được sử dụng thường xuyên trong C.

Và thường xuyên nhất, người ta dùng *return* để thông báo lỗi của chương trình: ví dụ, function trả về 1 (*true*) nếu tất cả diễn ra bình thường, và 0 (*false*) nếu có lỗi trong chương trình.

Một cách khác để sử dụng con trỏ trong function

Trong những code source mà chúng ta vừa thấy, không có con trỏ trong function *main*. Duy nhất chỉ biến số *soHang*.
Con trỏ duy nhất được sử dụng nằm trong function *triplePointer* (có *type int **)

Bạn cần biết rằng có cách viết khác cho đoạn code vừa rồi bằng cách thêm vào con trỏ trong function *main*:

C code:

```
void triplePointer(int *pointerSoHang);

int main (int argc, char *argv[ ])
{
   int soHang = 5;
   int *pointer = &soHang; // con tro nhan dia chi cua bien so soHang

   triplePointer (pointer); // Ta dua con tro (dia chi cua soHang) vao function
   printf ("%d", *pointer); // Ta hien thi gia tri cua soHang voi *pointer

   return 0;
}

void triplePointer(int *pointerSoHang)
{
   *pointerSoHang *= 3; // Ta x3 gia tri cua soHang
}
```

Hãy so sánh đoạn code source này với đoạn code source trước đó. Có một số thay đổi nhưng chúng sẽ cho ta cùng một kết quả:

Console

```
15
```

Điều cần xét đến là cách đưa địa chỉ của biến số *soHang* vào function, cách sử dụng địa chỉ của biến số *soHang*. Điều khác biệt xảy ra ở đây là cách tạo con trỏ trong function *main*.

VD trong *printf*, tôi muốn hiển thị giá trị của biến số *soHang* bằng cách viết **pointer*. Bạn cần biết rằng tôi vẫn thể viết *soHang*: kết quả sẽ giống nhau vì**pointer* và *soHang* đều có chung một giá trị trong bộ nhớ

Trong chương trình "Lớn hơn hay nhỏ hơn", chúng ta đã sử dụng con trỏ bất chấp việc biết nó là gì, trong việc sử dụng function *scanf*.

Thật ra, function này có tác dụng đọc những thông tin mà người dùng nhập vào bàn phím và gửi lại kết quả.

Để *scanf* có thể thay đổi trực tiếp giá trị của một biến số bằng cách nhập từ bàn phím, ta cần địa chỉ của biến số đó:

C code:

```
int soHang = 0;
scanf ("%d", &soHang);
```

function làm việc với con trỏ của biến số *soHang* và có thể thay đổi trực tiếp giá trị của *soHang*.

Và như chúng ta biết, chúng ta có thể làm như sau:

C code:

```
int soHang = 0;
int *pointer = &soHang;
scanf ("%d", pointer);
```

Chú ý là ta không đặt kí tự **&** trước pointer trong function *scanf* Tại đây, pointer bản thân nó đã là địa chỉ của biến số *soHang*, không cần thiết phải thêm **&** vào nữa !

Nếu bạn làm điều đó, bạn sẽ đưa cho *scanf* địa chỉ của pointer: nhưng thứ chúng ta cần là địa chỉ của *soHang*.

Giải quyết vấn đề nan giải ở đầu bài?

Đã đến lúc chúng ta xem lại tâm điểm của bài học. Nếu bạn hiểu bài học này, bạn đã có thể tự giải quyết vấn đề đặt ra. Hãy thử đi! trước khi xem kết quả tôi đưa bạn:

C code:

```
#include <stdio.h>
#include <stdlib.h>
void chuyenDoi(int *pointerGio, int *pointerPhut);

int main (int argc, char *argv[ ])
{
   int gio = 0, phut = 90;
   // Ta dua vao dia chi cua gio va phut
   chuyenDoi(&gio, &phut);
   // Luc nay, gia tri cua chung da duoc thay doi !
   printf ("%d gio va %d phut", gio, phut);
    return 0;
}

void chuyenDoi(int *pointerGio, int *pointerPhut)
{
   /*Note: dung quen dat dau * o phia truoc ten cua con tro! Bang cach nay ban co the thay doi gia tri cua bien so chu khong phai dia chi của no! Han la ban khong muon chia dia chi cua no dung khong? */
   *pointerGio = *pointerPhut / 60;
   *pointerPhut = *pointerPhut % 60;
}
```

Console:

```
1 gio va 30 phut
```

Không có gì khiến bạn ngạc nhiên trong đoạn code source này. Và như mọi khi, để tránh những nhầm lẫn không đáng có, tôi sẽ giải thích những gì đã diễn ra để chắc chắn rằng các bạn theo kịp tôi, vì đây là một bài học quan trọng, bạn cần cố gắng rất nhiều để hiểu, và tôi cũng cố gắng hết sức để giải thích rõ ràng giúp các bạn hiểu:

1. Biến số *gio* và *phut* được khởi tạo trong function *main*.

2. Ta gửi vào function *chuyenDoi* địa chỉ của *gio* và *phut*.

3. Function *chuyenDoi* nhận địa chỉ bằng cách đưa vào các con trỏ *pointerGio* và *pointerPhut*. Bạn cần biết rằng, cách gọi tên con trỏ không quan trọng. Tôi có thể gọi là *g* và *p*, hoặc cũng có thể là *gio* và *phut*.

4. function *chuyenDoi* thay đổi trực tiếp các giá trị của *gio* và *phut* trong bộ nhớ vì nó đã có địa chỉ của chúng trong các con trỏ. Và điều cần biết ở đây, tuyệt đối chấp hành, là phải đặt * trước tên của con trỏ nếu như ta muốn thay đổi giá trị của *gio* và *phut*. Nếu ta không làm việc này, ta sẽ thay đổi địa chỉ chứa trong con trỏ, và nó chẳng giúp ta được gì.

> Các bạn cần lưu ý là chúng ta vẫn có thể giải quyết "vấn đề" này không qua cách sử dụng con trỏ. Điều này là chắc chắn nhưng điều đó sẽ phá vỡ luật chúng ta đã đặt ra: không sử dụng những biến số *global*. Hoặc sử dụng *printf* trong function *chuyenDoi* (nhưng ta cần một *printf* trong function *main*).
>
> Mục đích chính của chương trình là giúp bạn có được hứng thú với việc sử dụng con trỏ. Và bạn hãy cố gắng thúc đẩy những hứng thú này ngày một nhiều hơn trong các bài học tiếp theo.
