[LẬP TRÌNH C/C++]

BIẾN CON TRỎ(CON TRỎ) CĂN BẢN

Một số lưu ý:

// <=> comment ghi chú, giải thích

; <=> kết thúc câu lệnh, luôn luôn có nếu không sẽ bị lỗi.

I. LỜI NÓI ĐẦU

- Như chúng ta đã biết thì các biến chính là các ô nhớ trong bộ nhớ máy tính(**RAM**), chúng ta có thể truy xuất đến các ô nhớ này thông qua việc gọi các tên biến.Các biến này được lưu trữ tại những chỗ cụ thể trong bộ nhớ.

- Đối với chương trình của chúng ta, bộ nhớ máy tính chỉ là 1 dãy gồm các ô nhớ 1 byte, mỗi ô có một địa chỉ xác định.

- Một mô hình dễ hiểu và thực tế về bộ nhớ máy tính chính là một phố trong một thành phố. Trên một phố tất cả các ngôi nhà đều được đánh số tuần tự với một cái tên duy nhất, nên nếu chúng ta nói đến số 50 Man Thiện thì chúng ta có thể tìm được nơi đó mà không lầm lẫn đi đâu được vì chỉ có một ngôi nhà với số như vậy.

- Cũng với cách tổ chức tương tự như việc đánh số các ngôi nhà, hệ điều hành tổ chức bộ nhớ thành những số đơn giản nhất, tuần tự, nên nếu chúng ta nói đến vị trí 6969 trong bộ nhớ, thì chúng ta biết chính xác ô nhớ đó vì chỉ có 1 vị trí với địa chỉ như vậy – địa chỉ duy nhất.

II. TOÁN TỬ LẤY ĐỊA CHỈ CỦA 1 BIẾN - &

- Vào thời điểm mà chúng ta khai báo ra 1 biến, thì biến đó phải được lưu trữ trong 1 vị trí cụ thể trong bộ nhớ. Nói chung thì chúng ta không thể nào quyết định được rằng nơi nào biến đó được lưu – công việc đó đã được làm tự động bởi trình biên dịch và hệ điều hành, nhưng một khi hệ điều hành đã gán một địa chỉ cho biến thì chúng ta hoàn toàn có thể biết được biến đó được lưu trữ ở đâu trong bộ nhớ.

- Điều này làm được là nhờ vào việc thực hiện cách đặt trước tên biến một dấu **&**(và), có nghĩa đây là *toán tử* " **lấy địa chỉ** "

VD:

```
a = 5;

b = a; // biến b sẽ giữ giá trị của biến a

c = &a;  // biến c sẽ giữ địa chỉ của biến a
```

Đây là ví dụ chỉ mang tính chất minh họa

- Toán tử địa chỉ &(và) có tác dụng chính là lấy địa chỉ của 1 biến từ bộ nhớ máy tính.

LƯU Ý:

- Những biến mà có thế lưu trữ được **địa chỉ** của một biến khác(như c) được gọi là **biến con trỏ**. Trong C++ **con trỏ** có rất nhiều ưu điểm và chúng được sử dụng rất thường xuyên.

Vậy con trỏ là gì ? Tại sao phải học và dùng con trỏ khi lập trình ? Ưu điểm và nhược điểm khi thao tác với con trỏ ?

Câu trả lời sẽ có trong các phần tiếp theo

III. TỔ CHỨC BỘ NHỚ TRONG MÁY TÍNH – TÌM HIỂU CƠ BẢN

- Trong bộ nhớ máy tính thường được phân chia thành 4 vùng nhớ chính như sau:

1. Code Segment

- Vùng nhớ này lưu trữ các mã máy dạng nhị phân. Các bạn có thể hình dung thế này – khi chúng ta viết chương trình trên các trình biên dịch(chương trình C, C++, Java, C#...), khi ta biên dịch để chạy chương trình đó, thì trình biên dịch sẽ chuyển các mã code của chúng ta thành các mã máy nhị phân – tức là vùng nhớ Code Segment này chỉ bao gồm các số 0 và 1.

2. Data Segment

- Vùng nhớ này chứa các *biến tĩnh(static)* và các *biến toàn cục*(vượt ra ngoài phạm vi các hàm).

VD: Biến toàn cục

```
#include<iostream>
```

```
using namespace std;


int a = 69;
// biến a được gọi là biến toàn cục - có phạm vi sử dụng trong toàn bộ chương trình
// - tính từ vị trí khai báo, khởi tạo đến cuối chương trình

int Tim_Max(int x, int y)
{
	return x > y ? x : y;
}

int main()
{
	cout << Tim_Max(5, 10);

	system("pause");
	return 0;
}
```

3. Heap Segment

- *Heap* là vùng nhớ thuộc phân khúc cho người lập trình(bạn) **tự do cấp phát và sử dụng**. Tuy nhiên lại **không do CPU quản lí**, vì vậy người lập trình phải tự mình quản lí vùng nhớ này khi lập trình. Nếu người lập trình sử dụng vùng nhớ này mà không tuân theo cơ chế *có cấp phát – phải giải phóng*, vô tình bỏ sót thì sẽ dẫn đến hiện tượng memory leak(khi nào lớn sẽ biết).

- Vùng nhớ này thường dùng cho việc cấp phát bộ nhớ cho **con trỏ**.

- Vùng nhớ "Heap" là vùng nhớ mang tính chất toàn cục, không phụ thuộc vào một lần gọi hàm nào cả. Với công nghệ lập trình C/C++, sau khi xin cấp phát bộ nhớ từ "Heap" và sử dụng xong thì người lập trình cần phải chủ động gọi hàm giải phóng vùng nhớ.

4. Stack Segment

- Vùng nhớ này có thể được hiểu theo cơ chế **Fisrt In - Last Out(LIFO)**. Đây là vùng nhớ do CPU quản lí, lưu trữ các biến cục bộ trong chương trình, người lập trình không thể nào can thiệp vào vùng nhớ này – nếu cố tình can thiệp sẽ gây ra lỗi khá nghiêm trọng.

- **Stack** lưu trữ các **biến cục bộ** - nghĩa là các biến này là các biến được khai báo tạm thời và sử dụng trong các hàm(phương thức) – kể cả hàm **main().** Đơn giản là những biến nào được khai báo bên trong cặp ngoặc **{}** thì được gọi là **biến cục bộ** - được sử dụng tạm thời.

- Khi chúng ta gọi 1 hàm thì các biến bên trong hàm đó sẽ được đưa vào Stack để thực thi, khi kết thúc hàm đó thì các biến cục bộ bên trong hàm sẽ được **CPU tự giải phóng.**

VD:

```
#include<iostream>
using namespace std;

// hàm hoán vị 2 số nguyên cho nhau
void Hoan_Vi(int &x, int &y)
{
	int temp; // biến temp là biến cục bộ thuộc vùng nhớ Stack Segment
	temp = x;
	x = y;
	y = temp;
}

int main()
{
	// 2 biến x và y đều là biến cục bộ thuộc vùng nhớ Stack Segment
	int x = 9;
	int y = 69;

	system("pause");
	return 0;
}
```

IV. BIẾN CON TRỎ

1. Khái niệm biến con trỏ

- Biến **con trỏ** dùng để **lưu địa chỉ** của những **biến khác** hay địa chỉ 1 vùng nhớ hợp lệ do người lập trình cấp phát. Thông thường nhất là vùng nhớ được cấp phát nhờ gọi các hàm có sẵn trong thư viện để xin cấp phát vùng nhớ từ hệ điều hành.

- *Các hàm có sẵn hỗ trợ việc cấp phát bộ nhớ cho con trỏ trong ngôn ngữ lập trình C:* **malloc, calloc, realloc**

- *Trong ngôn ngữ lập trình C++ để cấp phát bộ nhớ cho con trỏ thì ta dùng toán tử* **new**

2. Sự khác nhau giữa biến thường và biến con trỏ

- **Biến bình thường** gồm có 2 thành phần là:

+ **Địa chỉ của biến**: khi khai báo biến để sử dụng thì trình biên dịch và hệ điều hành sẽ cấp phát 1 nơi nào đó trong bộ nhớ để lưu trữ biến – nơi đó gọi là địa chỉ của biến

+ **Giá trị của biến**: chính là nơi lưu giữ giá trị của biến sau khi khởi tạo

- **Biến con trỏ** cũng gồm 2 thành phần cơ bản như biến bình thường, nhưng biến con trỏ có thêm 1 *tính chất ưu việt hơn so với biến bình thường*, đây được xem là niềm kiêu hãnh của biến con trỏ - đó chính là **miền giá trị** của con trỏ.

Miền giá trị là gì ?

Miền giá trị của con trỏ chính là **địa chỉ** của biến mà con trỏ đang trỏ tới.

3. Khai báo biến kiểu con trỏ

Cú pháp

<kiểu dữ liệu trỏ tới> *<tên biến>;

Trong đó:

- **kiểu dữ liệu trỏ tới**: đây không phải là kiểu dữ liệu của biến con trỏ mà là kiểu dữ liệu của cái biến mà con trỏ sẽ trỏ tới.

- **dấu *** đặt trước cái biến về cơ bản là quy ước muốn thể hiện biến đó là biến con trỏ, phân biệt với các biến bình thường khác.

VD:

```
int a = 69; // khởi tạo biến số nguyên a có giá trị là 69
int *b; // khai báo ra 1 biến con trỏ - chỉ có thể trỏ tới các biến thuộc kiểu dữ liệu số nguyên
```

- Chúng ta muốn truy xuất đến giá trị của biến con trỏ thì dùng toán tử *

4. Con trỏ - trỏ tới biến

Cách 1:

```
int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
int *b; // khai báo biến con trỏ b, có kiểu dữ liệu con trỏ là số nguyên - nghĩa là con trỏ b chỉ có thể trỏ đến các biến số nguyên để lấy địa chỉ
b = &a; // câu lệnh cho phép con trỏ b trỏ đến biến a để lấy địa chỉ của biến a - thông qua toán tử địa chỉ &
```

Cách 2:

```
int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
int *b = &a; // ý nghĩa giống cách 1 - nhưng đây là cách làm gộp
```

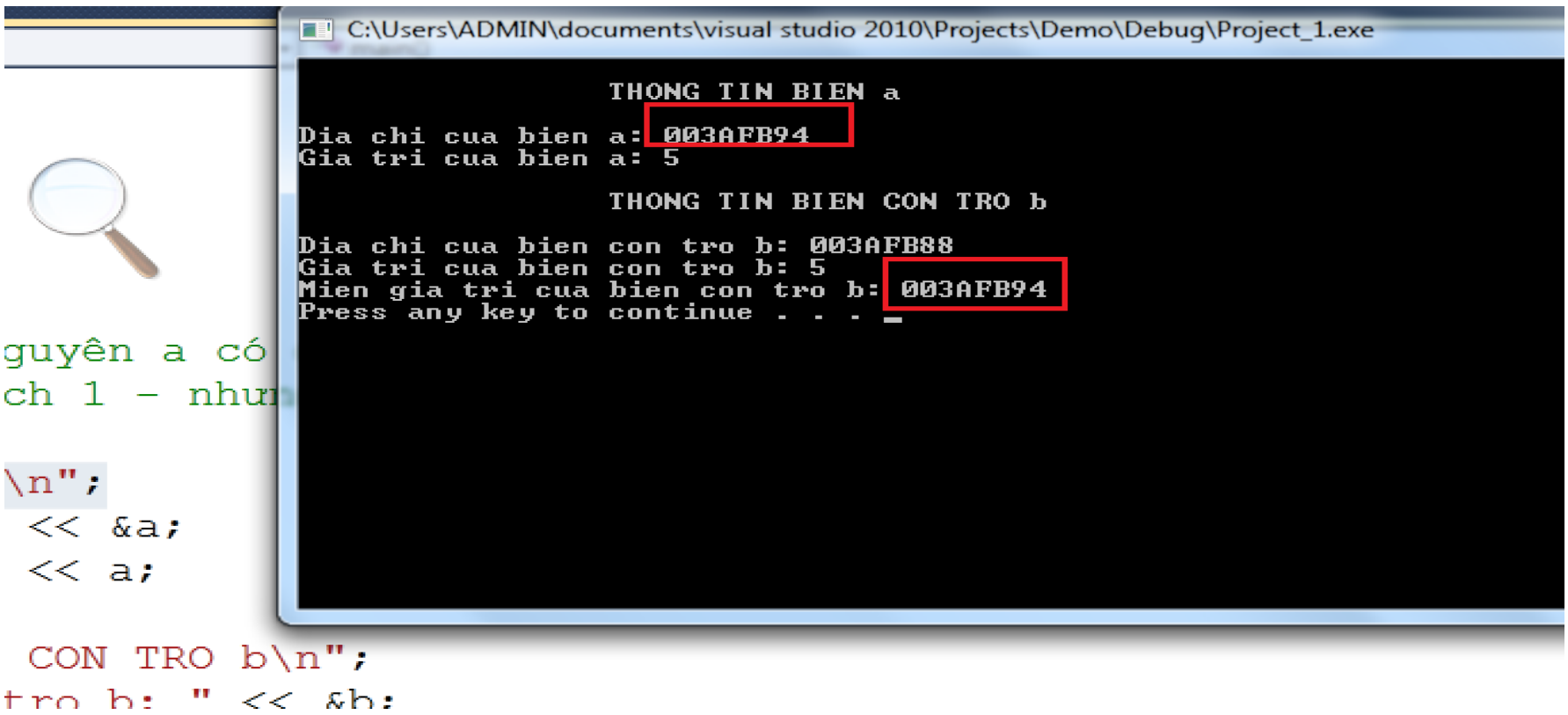
Nhận xét: b = &a

- Trước biến **a** phải có toán tử địa chỉ **&** để cho 2 *vế trái và phải đồng nhất về mặt ý nghĩa*. Bên trái - **b** đang là 1 con trỏ, mà bản chất của con trỏ là **lưu trữ địa chỉ**, vì vậy vế phải cũng phải là 1 **địa chỉ** - thông qua toán tử **&** để lấy địa chỉ của biến **a** gán qua cho con trỏ **b.**

- Bản chất của con trỏ là lưu trữ địa chỉ của 1 biến khác. Miền giá trị của con trỏ sẽ lưu địa chỉ của biến mà con trỏ sẽ trỏ đến.

Code demo

```
#include<iostream>
using namespace std;


int main()
{
	int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
	int *b = &a; // ý nghĩa giống cách 1 - nhưng đây là cách làm gộp

	cout << "\n\t\t THONG TIN BIEN a\n";
	cout << "\nDia chi cua bien a: " << &a;
	cout << "\nGia tri cua bien a: " << a;

	cout << "\n\n\t\t THONG TIN BIEN CON TRO b\n";
	cout << "\nDia chi cua bien con tro b: " << &b;
	cout << "\nGia tri cua bien con tro b: " << *b;
	cout << "\nMien gia tri cua bien con tro b: " << b;
	cout << "\n";
	system("pause");
	return 0;
}
```

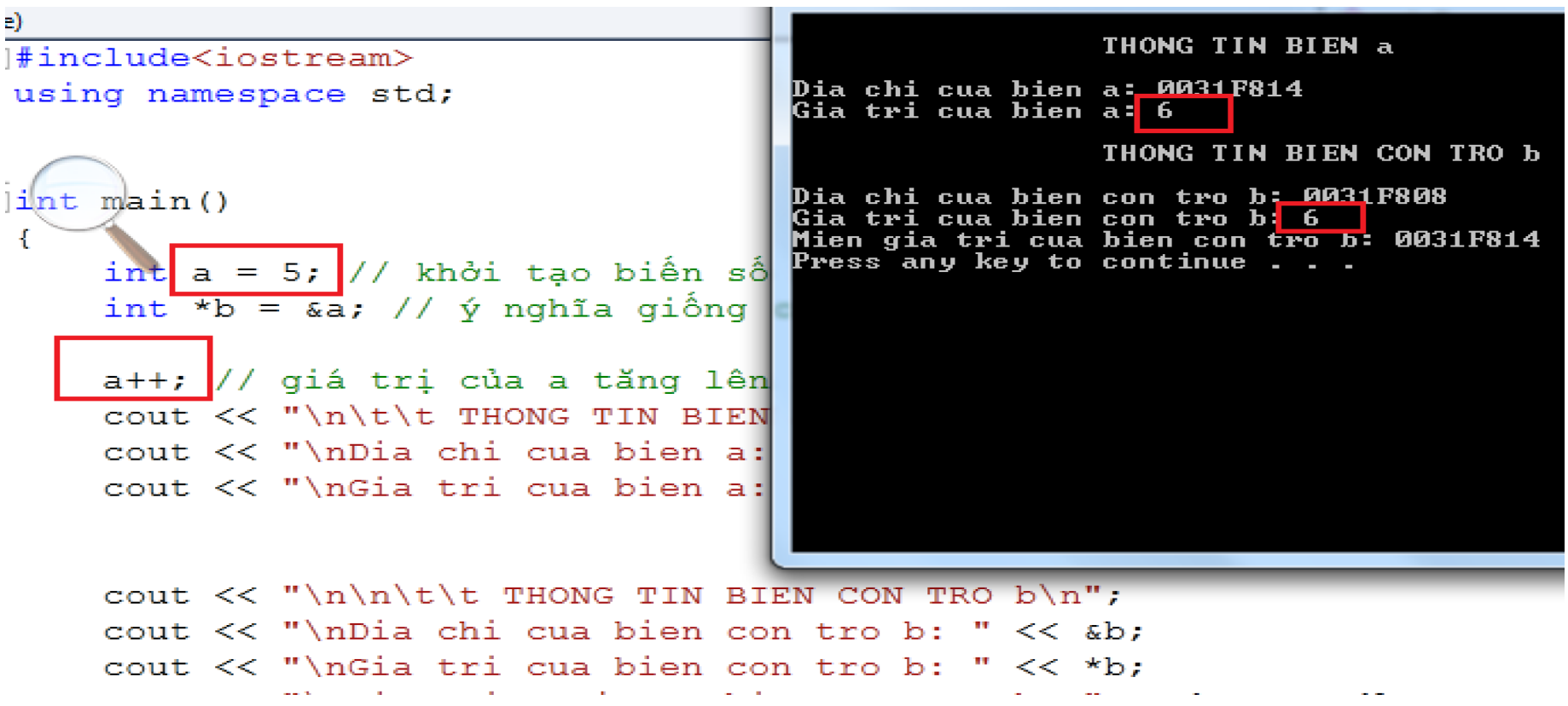


- Khi chúng ta cho **con trỏ b** trỏ đến **biến a** thì giá trị của **biến con trỏ b** chính là giá trị của **biến a**. Nếu 1 trong 2 biến a hay b **thay đổi giá trị** thì **biến còn lại cũng sẽ thay đổi giá trị** theo. Bởi vì **a** và **b** đã tạo **mối liên kết** với nhau thông qua lệnh b = &a.

Code demo

```
#include<iostream>
using namespace std;


int main()
{
	int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
	int *b = &a; // ý nghĩa giống cách 1 - nhưng đây là cách làm gộp

	a++; // giá trị của a tăng lên 1 đơn vị
	cout << "\n\t\t THONG TIN BIEN a\n";
	cout << "\nDia chi cua bien a: " << &a;
```

```
        cout << "\nGia tri cua bien a: " << a;


        cout << "\n\n\t\t THONG TIN BIEN CON TRO b\n";
        cout << "\nDia chi cua bien con tro b: " << &b;
        cout << "\nGia tri cua bien con tro b: " << *b;
        cout << "\nMien gia tri cua bien con tro b: " << b << endl;


        system("pause");
        return 0;
}
```

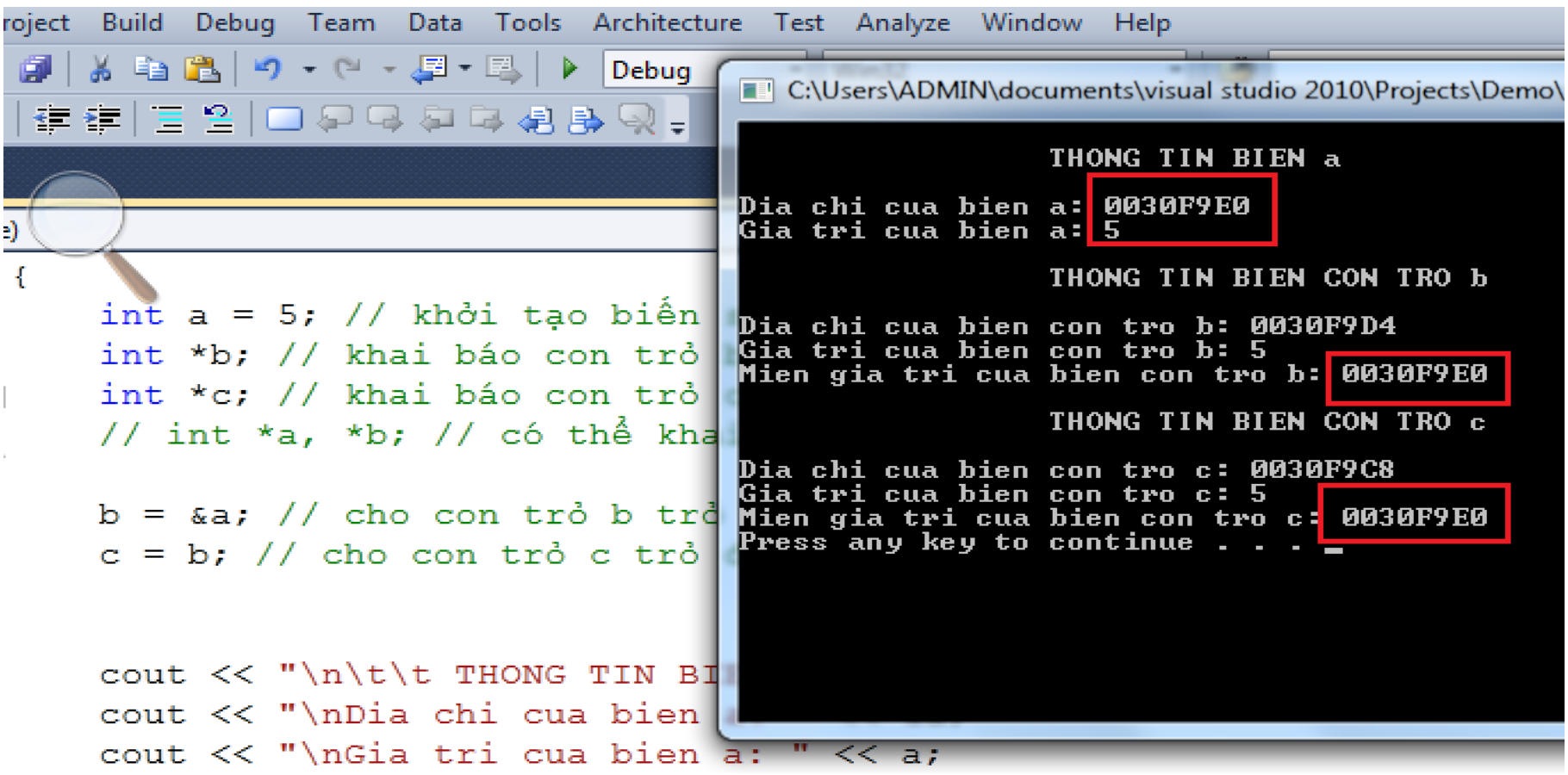


5. Con trỏ - trỏ tới con trỏ

Code demo 1

```
#include<iostream>
using namespace std;
int main()
{
        int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
        int *b; // khai báo con trỏ b
        int *c; // khai báo con trỏ c
        // int *a, *b; // có thể khai báo gộp

        b = &a; // cho con trỏ b trỏ đến biến a
        c = b; // cho con trỏ c trỏ đến con trỏ b, lúc này vì là 2 con trỏ nên vế trái và phải đồng nhất về mặt ý nghĩa


        cout << "\n\t\t THONG TIN BIEN a\n";
        cout << "\nDia chi cua bien a: " << &a;
        cout << "\nGia tri cua bien a: " << a;


        cout << "\n\n\t\t THONG TIN BIEN CON TRO b\n";
        cout << "\nDia chi cua bien con tro b: " << &b;
        cout << "\nGia tri cua bien con tro b: " << *b;
        cout << "\nMien gia tri cua bien con tro b: " << b;
```

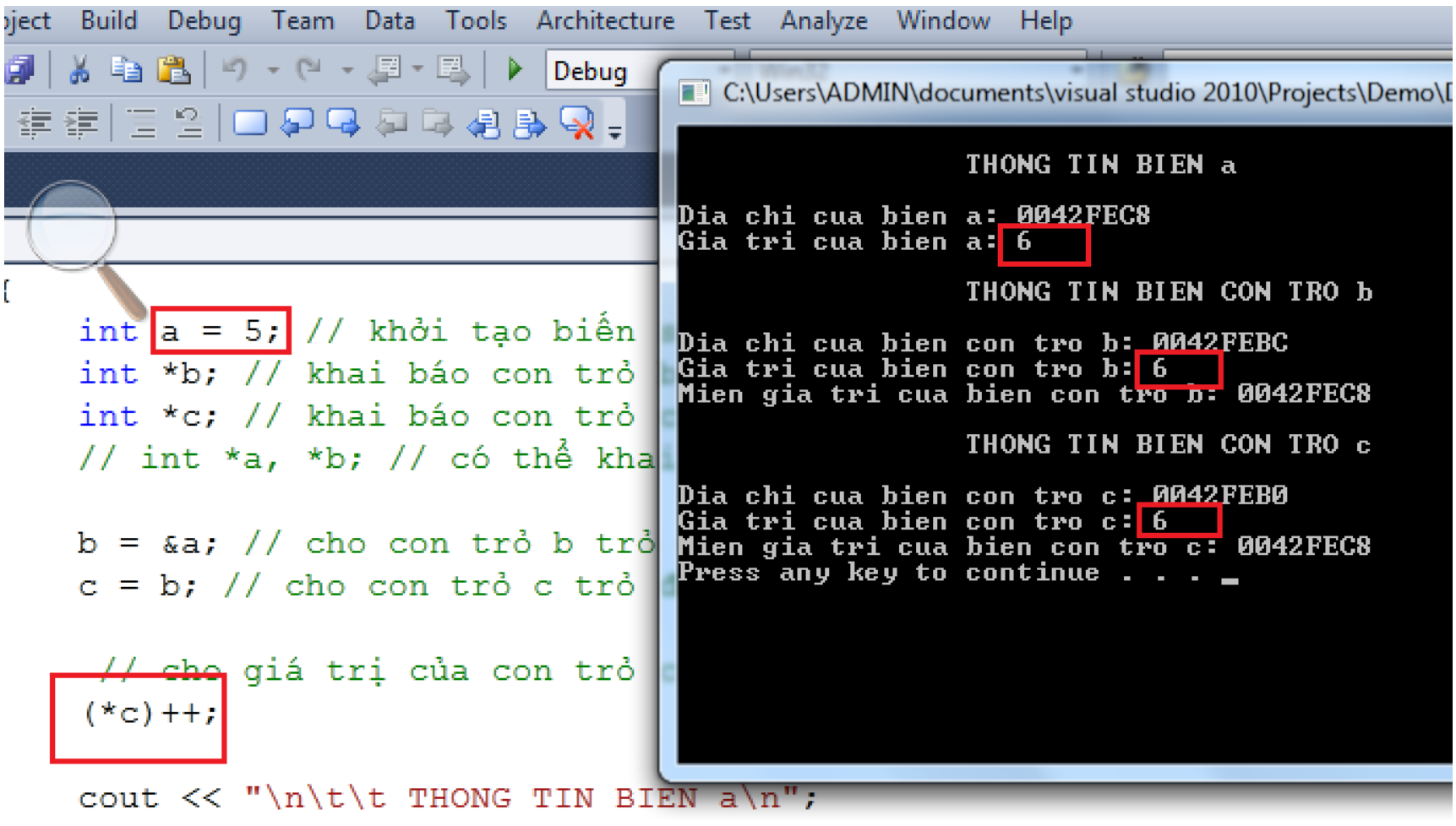
```
        cout << "\n\n\t\t THONG TIN BIEN CON TRO c\n";
        cout << "\nDia chi cua bien con tro c: " << &c;
        cout << "\nGia tri cua bien con tro c: " << *c;
        cout << "\nMien gia tri cua bien con tro c: " << c << endl;

        system("pause");
        return 0;
}
```



Code demo 2

```
#include<iostream>
using namespace std;


int main()
{
        int a = 5; // khởi tạo biến số nguyên a có giá trị là 5
        int *b; // khai báo con trỏ b
        int *c; // khai báo con trỏ c
        // int *a, *b; // có thể khai báo gộp

        b = &a; // cho con trỏ b trỏ đến biến a
        c = b; // cho con trỏ c trỏ đến con trỏ b, lúc này vì là 2 con trỏ nên vế trái và phải đồng nhất về mặt ý nghĩa

         // cho giá trị của con trỏ c tăng lên 1 đơn vị
        (*c)++;

        cout << "\n\t\t THONG TIN BIEN a\n";
        cout << "\nDia chi cua bien a: " << &a;
        cout << "\nGia tri cua bien a: " << a;


        cout << "\n\n\t\t THONG TIN BIEN CON TRO b\n";
        cout << "\nDia chi cua bien con tro b: " << &b;
        cout << "\nGia tri cua bien con tro b: " << *b;
        cout << "\nMien gia tri cua bien con tro b: " << b;
```

```
        cout << "\n\n\t\t THONG TIN BIEN CON TRO c\n";
        cout << "\nDia chi cua bien con tro c: " << &c;
        cout << "\nGia tri cua bien con tro c: " << *c;
        cout << "\nMien gia tri cua bien con tro c: " << c << endl;

        system("pause");
        return 0;
}
```

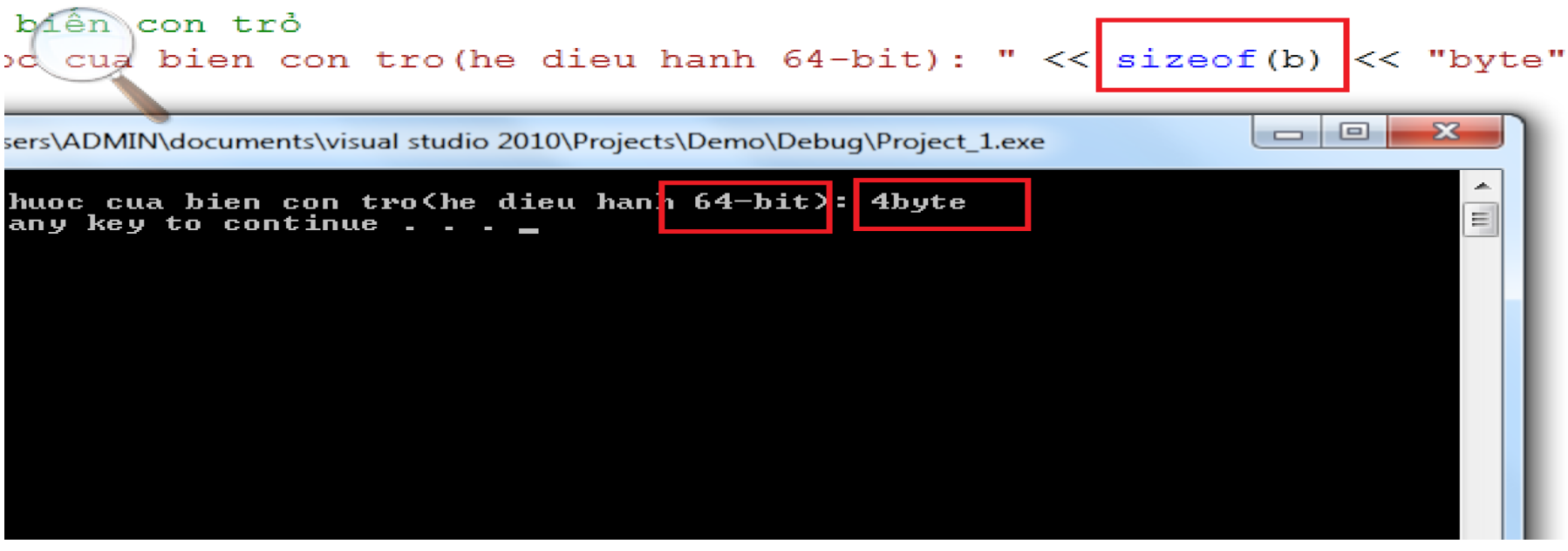


6. Kích thước – độ lớn của con trỏ

- Kích thước của biến con trỏ trong hệ điều hành Microsoft windows 32-bit là 4 byte cho mọi kiểu dữ liệu

- Kích thước của biến con trỏ trong hệ điều hành Microsoft windows 64-bit là 8 byte cho mọi kiểu dữ liệu

- Chúng ta có thể dùng hàm **sizeof()** để kiểm tra kích thước của kiểu dữ liệu con trỏ.



Text độ lớn con trỏ trong hệ điều hành Windows 64bit

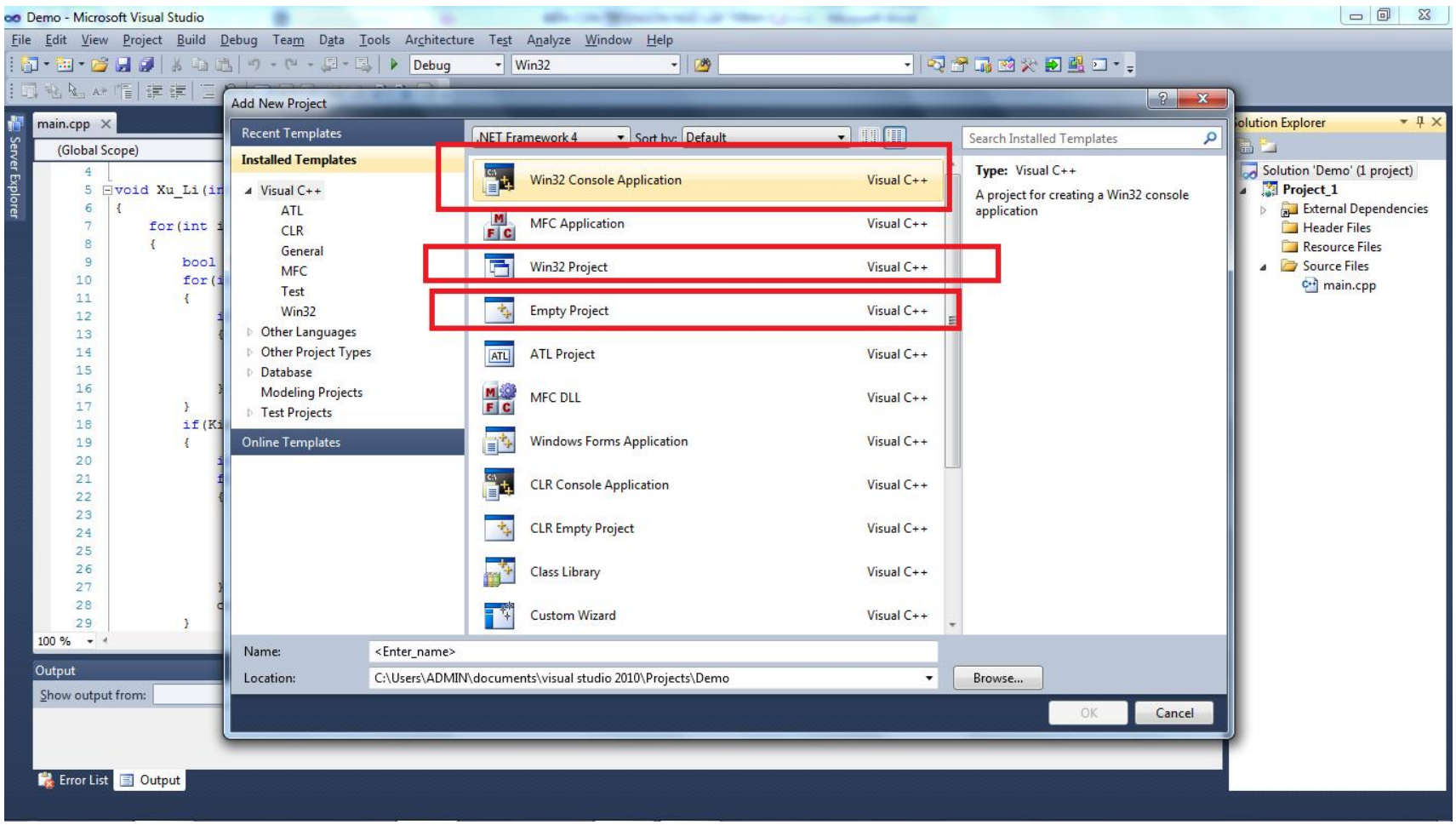
- Qua demo ở trên các bạn thấy rằng kích thước của con trỏ trong hệ điều hành Microsoft Windows (64-bit) lại chỉ là 4 byte ?

Có gì đó sai sai ở đây chăng ?

Chú ý:

- Kích thước – độ lớn của kiểu dữ liệu con trỏ nó còn phụ thuộc vào trình biên dịch và hệ điều hành hiện tại mà ta đang sử dụng.

- Trường hợp tôi test trên hệ điều hành Microsoft Windows (64-bit) lại chỉ là 4 byte, bởi vì khi tôi tạo Solution hay Project trong Visual Studio để quản lí code thì trình biên dịch mặc định là Win32 nên độ lớn của con trỏ luôn luôn là 4 byte.



Visual Studio mặc định Win32

7. Con trỏ vô kiểu – void*

- Con trỏ vô kiểu **void*** là 1 con trỏ có thể nhận mọi kiểu dữ liệu mà nó trỏ tới– nghĩa là con trỏ **void*** có thể trỏ đến mọi kiểu dữ liệu mà nó muốn, nhưng người lập trình cần *phải ép kiểu dữ liệu* để phù hợp với kiểu mà con trỏ trỏ tới.

Khái niệm void là gì khó hiểu quá ? ép kiểu dữ liệu là gì càng mù mờ thêm ? Bạn sẽ hiểu hơn thông qua demo*

Demo_1: void trỏ đến biến int*

```
#include<iostream>
using namespace std;

int main()
{
	int a = 69; // khởi tạo biến số nguyên a có giá trị là 69
	void* b; //<=> void *b; khai báo biến con trỏ vô kiểu void* - nghĩa là b có thể trỏ đến mọi kiểu dữ liệu mà nó muốn
	b = &a; // cho b trỏ đến biến a - nghĩa là b đang trỏ đến địa chỉ của biến a để lấy địa chỉ của biến a

	cout << "\nGia tri cua con tro b: " << *(int *)b; // thường thì ta để *p - xuất ra giá trị của con trỏ b, nhưng b lúc này đang là con trỏ vô kiểu void* nên ta phải ép kiểu - kiểu dữ liệu ép kiểu chính là kiểu dữ liệu mà con trỏ b đó trỏ tới

	system("pause");
	return 0;
}
```

Demo_2: void trỏ đến biến double*

```
#include<iostream>
using namespace std;

int main()
{
	double a = 69; // khởi tạo biến số thực a có giá trị là 69
	void* b; //<=> void *b; khai báo biến con trỏ vô kiểu void* - nghĩa là b có thể trỏ đến mọi kiểu dữ liệu mà nó muốn
	b = &a; // cho b trỏ đến biến a - nghĩa là b đang trỏ đến địa chỉ của biến a để lấy địa chỉ của biến a

	cout << "\nGia tri cua con tro b: " << *(double *)b; // thường thì ta để *p - xuất ra giá trị của con trỏ b, nhưng b lúc này đang là con trỏ vô kiểu void* nên ta phải ép kiểu - kiểu dữ liệu ép kiểu chính là kiểu dữ liệu mà con trỏ b đó trỏ tới

	system("pause");
	return 0;
}
```

V. CÁC CƠ CHẾ CẤP PHÁT BỘ NHỚ VÀ GIẢI PHÓNG CHO CON TRỎ

1. Trong ngôn ngữ lập trình C

- Ta có các cơ chế **cấp phát vùng nhớ** cho con trỏ thông qua các hàm:

	+ malloc

	+ calloc

	+ realloc

- Cơ chế **giải phóng vùng nhớ** cho con trỏ sau khi sử dụng:

	+ free

HÀM CẤP PHÁT – GIẢI PHÓNG BỘ NHỚ CHO CON TRỎ		
Hàm	**Cú pháp sử dụng**	**Mô tả ý nghĩa hàm**
malloc	void *malloc(int size)	- Cấp phát 1 vùng bộ nhớ từ **heap** với số **byte** là **size** – kích thước bộ nhớ cần cấp phát(**byte**). Hàm trả về địa chỉ vùng nhớ được cấp phát , được gán vào con trỏ khi gọi hàm. Nếu hệ điều hành không đủ bộ nhớ thì hàm trả về **NULL**.
calloc	void *calloc(int count, int size)	- Cấp phát bộ nhớ từ **heap** và khởi tạo giá trị 0 cho tất cả các phần tử **count** trong vùng nhớ cần cấp phát. Hàm này xin cấp phát **count**(*count chính là số lượng phần tử cần cấp phát*) phần tử liên tục nhau trong bộ nhớ, mỗi phần tử có kích thước là **size(byte)** -như vậy tổng số byte được cấp phát là **count * size**. Giá trị trả về như hàm **malloc()**
realloc	void *realloc(void *p, int size)	*TH1: Đối với vùng nhớ chưa được khởi tạo thì realloc sẽ khởi tạo vùng nhớ mới. *TH2: Nếu vùng nhớ đã tồn tại thì *realloc* có chức năng thu hẹp lại (giảm) hay giãn ra thêm (tăng) số lượng phần tử theo yêu cầu của lập trình viên. - Đối với hàm này thì tham số **p** là 1 con trỏ giữ địa chỉ của vùng nhớ yêu cầu trình biên dịch và hệ điều hành cấp phát. - **size** là kích thước**(byte)** của vùng nhớ muốn cấp phát
free	void free(void *p)	- Giải phóng vùng nhớ đã cấp phát có địa chỉ được chứa trong con trỏ **p**. Chú ý rằng biến **p** phải chứa địa chỉ hợp lệ của vùng nhớ trong **heap** đã được cấp, chưa được giải phóng lần nào. - Việc giải phóng bản chất là thông báo cho hệ điều hành biết rằng: vùng nhớ vừa cấp phát sẽ không sử dụng nữa, nên hệ điều hành thấy ứng dụng, tiến trình nào cần vùng nhớ, thì có thể lấy để cấp cho tiến trình, ứng dụng tương ứng.

Lưu ý:

- Khi sử dụng các hàm cấp phát vùng nhớ cho con trỏ trong ngôn ngữ lập trình C, thì chúng ta cần phải khai báo thư viện <stdlib.h>

a. Hiện thực hàm malloc

```
#include<stdio.h>
```

```c
#include<conio.h>
#include<stdlib.h> // khai báo thư viện stdlib.h để có thể sử dụng các hàm malloc, calloc, realloc

int main()
{
	int *a; // khai báo con trỏ a có kiểu dữ liệu là int
	// cấp phát vùng nhớ cho con trỏ a kiểu số nguyên để chứa giá trị
	a = (int *)malloc(sizeof(int));

	float *b; // khai báo con trỏ b có kiểu dữ liệu là float
	// cấp phát vùng nhớ cho con trỏ b kiểu số thực để chứa giá trị
	b = (float *)malloc(sizeof(float));

	free(a); // giải phóng vùng nhớ cho con trỏ a
	free(b); // giải phóng vùng nhớ cho con trỏ b
	return 0;
}
```

b. Hiện thực hàm calloc

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> // khai báo thư viện stdlib.h để có thể sử dụng các hàm malloc, calloc, realloc

int main()
{
	int *a; // khai báo con trỏ a có kiểu dữ liệu là int
	// cấp phát vùng nhớ cho con trỏ a kiểu số nguyên để chứa giá trị
	a = (int *)calloc(1, sizeof(int));

	float *b; // khai báo con trỏ b có kiểu dữ liệu là float
	// cấp phát vùng nhớ cho con trỏ b kiểu số thực để chứa giá trị
	b = (float *)calloc(1, sizeof(float));

	free(a); // giải phóng vùng nhớ cho con trỏ a
	free(b); // giải phóng vùng nhớ cho con trỏ b
	return 0;
}
```

c. Hiện thực hàm realloc

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> // khai báo thư viện stdlib.h để có thể sử dụng các hàm malloc, calloc, realloc

int main()
{
	int *a; // khai báo con trỏ a có kiểu dữ liệu là int
	// cấp phát vùng nhớ cho con trỏ a kiểu số nguyên để chứa giá trị
	a = (int *)realloc(NULL, sizeof(int));

	float *b; // khai báo con trỏ b có kiểu dữ liệu là float
	// cấp phát vùng nhớ cho con trỏ b kiểu số thực để chứa giá trị
	b = (float *)realloc(NULL, sizeof(float));

	free(a); // giải phóng vùng nhớ cho con trỏ a
	free(b); // giải phóng vùng nhớ cho con trỏ b
	return 0;
}
```

2. Trong ngôn ngữ lập trình C++

- Ta có cơ chế **cấp phát bộ nhớ** cho con trỏ thông qua toán tử **new**

Cú pháp:

<kiểu dữ liệu trỏ tới> *<tên con trỏ> = new <kiểu dữ liệu trỏ tới>[<số lượng phần tử cần cấp phát>];

```
#include<iostream>
using namespace std;

int main()
{
	int *a;
	// cấp phát vùng nhớ cho biến con trỏ a để chứa giá trị là số nguyên
	a = new int; // <=> a = new int[1] khai báo ra vùng nhớ cho con trỏ, vì chỉ cần 1 ô nhớ để chứa con trỏ nên chỉ cần new int

	// cấp phát vùng nhớ cho biến con trỏ b để chứa giá trị là số thực
	float *b = new float; // có thể làm gộp

	system("pause"); // dừng màn hình để xem
	return 0;
}
```

Nhận xét:

- Khi chúng ta khai báo ra biến con trỏ- thì con trỏ lúc này chỉ tồn tại duy nhất là địa chỉ trong bộ nhớ máy tính – chưa hề tồn tại vùng nhớ để chứa giá trị. Vì vậy nếu chúng ta muốn con trỏ đó có thể chứa giá trị thì ta cần phải cấp phát vùng nhớ cho nó.

- Cơ chế làm việc của các hàm malloc, calloc, realloc và toán rử new như sau:

+ Khi trình biên dịch gặp các hàm cấp phát vùng nhớ cho con trỏ như malloc, calloc, realloc và toán tử new thì trình biên dịch sẽ yêu cầu hệ điều hành cấp phát ra 1 vùng nhớ trong bộ nhớ máy tính – cụ thể là trong vùng nhớ HEAP – vùng nhớ do người lập trình quản lí – CPU không quản lí, sau khi vùng nhớ cấp phát đã tồn tại(vùng nhớ hợp lệ) thì biến con trỏ sẽ trỏ đến địa chỉ của vùng nhớ đó. Lúc này biến con trỏ đã có vùng nhớ cho riêng mình để chứa giá trị.

- Khi chúng ta cấp phát vùng nhớ cho con trỏ để sử dụng, vì các hàm malloc, calloc, realloc và toán tử new là các hàm, toán tử yêu cầu hệ điều hành cấp phát vùng nhớ - trong vùng nhớ HEAP của máy tính. Mà vùng nhớ HEAP là vùng nhớ do người lập trình quản lí – CPU không quản lí. Vì vậy khi chúng ta cấp phát vùng nhớ cho con trỏ, thì sau khi không dùng đến con trỏ nữa - chúng ta phải có cơ chế giải phóng vùng nhớ đó – dùng hàm free(đối với ngôn ngữ lập trình C) – dùng hàm delete(đối với ngôn ngữ lập trình C++). Nếu không giải phóng thì vùng nhớ đó vẫn sẽ tồn tại – nếu chúng ta cấp phát nhiều con trỏ mà không giải phóng thì sẽ gây ra tình trạng tràn bộ nhớ.

- Luôn luôn nhớ thần chú " Có cấp phát – phải giải phóng ".

- Việc giải phóng vùng nhớ cho con trỏ trong ngôn ngữ lập trình C/C++ thực chất là thông báo cho hệ điều hành biết rằng: vùng nhớ mà hệ điều hành vừa cấp phát cho con trỏ dùng - bây giờ không cần dùng nữa – hệ điều hành xem chương trình hay là tiến trình nào cần thì cấp phát cho chương trình hay tiến trình tương ứng đó.

---------- ***END*** ----------

- Đây là tài liệu dùng để học và tham khảo.

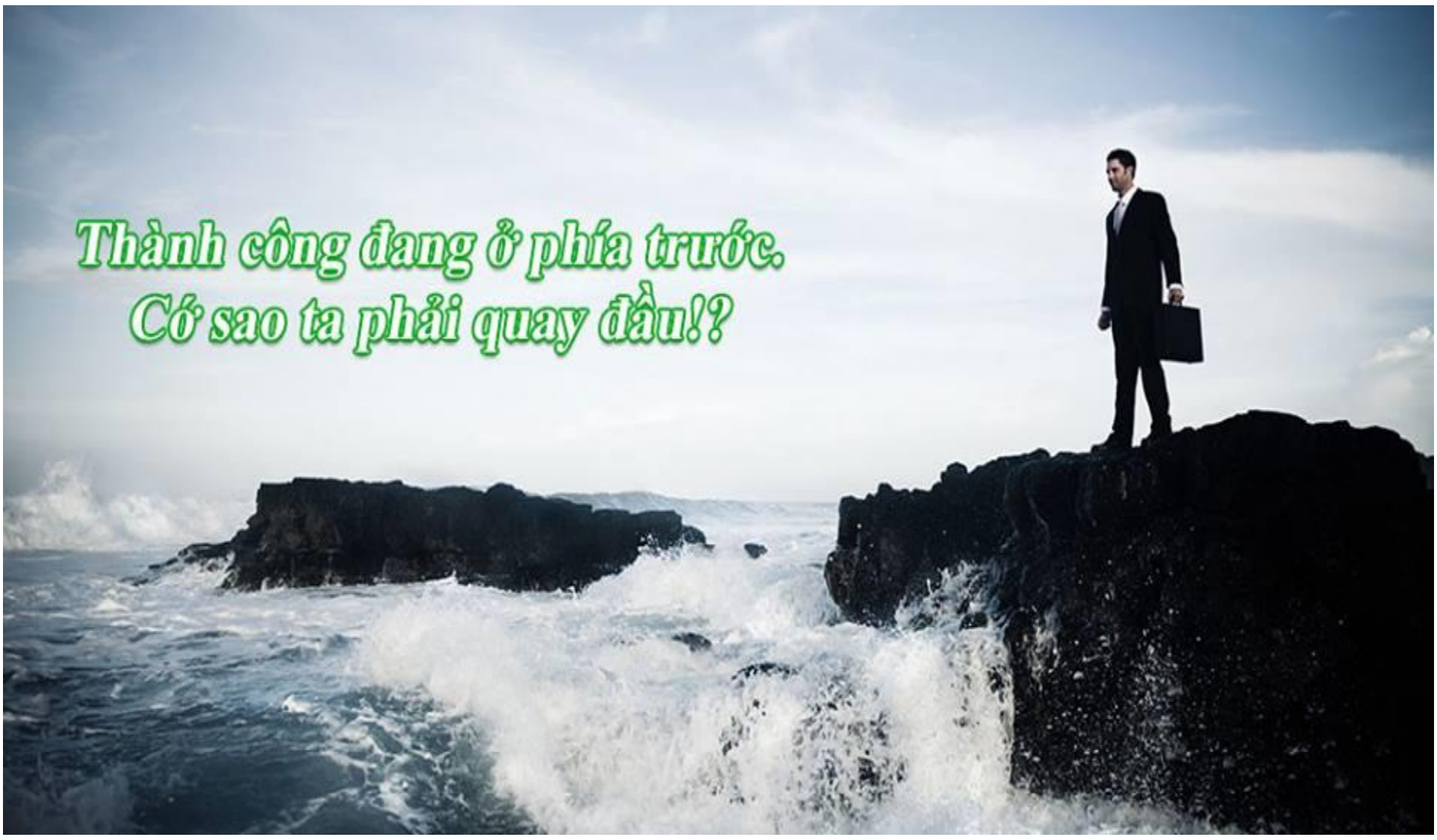
- Trong quá trình biên soạn bộ tài liệu có:

+ Tham khảo 1 số nguồn tài liệu(google search, ebook, sách chuyên ngành…).

+ Kiến thức – kinh nghiệm cá nhân của tôi.

- Có thể còn 1 số lỗi sai sót – sẽ luôn cập nhật phiên bản mới.

- Mong nhận được sự đóng góp và phản hồi tích cực từ các bạn đọc. Chân thành cảm ơn.


Thành công đang ở phía trước.
Có sao ta phải quay đầu!?