

# **Nội dung 8**

## **Con trỏ - Pointer và xâu ký tự**

**TS. Trần Thanh Hải**

# Nội dung

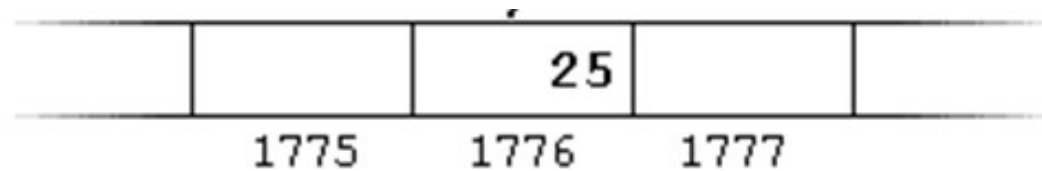
---

- Giới thiệu.
- Khái niệm con trỏ.
- Các toán tử con trỏ.
- Gán giá trị thông qua con trỏ.
- .....
- Bộ nhớ động.

## Giới thiệu

---

- **Biến là:** các ô nhớ trong bộ nhớ của máy tính và có thể được truy cập bằng các định danh (tên của biến).
- Bộ nhớ của máy tính giống như các ô (cell ) nhớ liên tiếp, tính theo byte, và có địa chỉ là duy nhất.
- **Ví dụ:** ô nhớ với địa chỉ 1776 luôn luôn theo sau đó là ô nhớ địa chỉ 1775 và trước nó là 1777 và cách sau 1000 là ô nhớ 776 và cách trước 1000 là 2776.



- Khi đó hệ thống cung cấp một vị trí cho biến trong bộ nhớ để lưu biến.
- Theo cách này chương trình không quan tâm đến địa chỉ vật lý của dữ liệu trong bộ nhớ. Sử dụng đơn giản các biến khi cần tham chiếu đến biến.

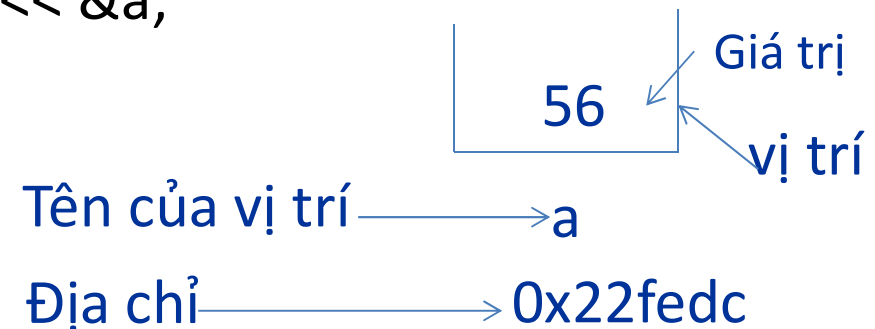
## Giới thiệu

- Ví dụ chương trình in địa chỉ của biến. [Lt81\\_pointer\\_m dau](#)

```
int a = 56;
```

```
cout<< "Gia tri cua a = "<<a; // truy cập bằng định danh.
```

```
cout<< "Dia chi cua bien a = "<< &a;
```



```
E:\tincoso4\side\noidung7\noidung7_poin
Gia tri cua a = 56
Dia chi cua bien a = 0x22fedc
Press any key to continue . . .
```

## Thảo luận

---

- Vậy có **biến** nào có thể chứa được địa chỉ của biến khác không? Nếu có thì:
- Lợi ích gì của việc sử dụng biến này.
- Khai báo như thế nào? Cách lưu địa chỉ và cách lấy giá trị?. Gán giá trị ban đầu như thế nào
- Mỗi quan hệ với các biến khác như thế nào.
- Các phép toán nào liên quan đến nó.
- Mỗi liên hệ giữa mảng và biến này như thế nào?
- Cấp phát động như thế nào. Vị trí biến này được lưu ở đâu trong bộ nhớ....

# 1. Con trỏ là gì?

---

- **Vấn đề:** ta có khai báo sau

```
float y = 10.4; // name – type – value
```

```
cout<< “Dia chi cua bien y”<< &y;
```

Như vậy, biến y sẽ chứa các giá trị kiểu float. (10.4)

Và tham chiếu đến nó sử dụng tên biến y.

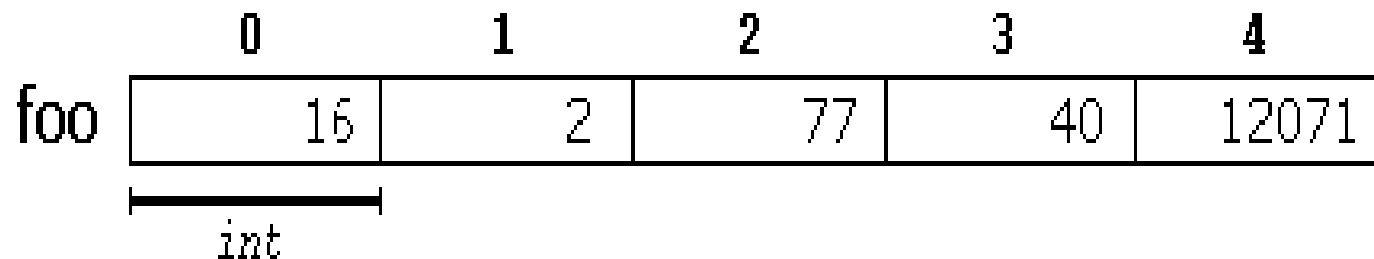
Vậy muốn lưu trữ địa chỉ của biến &y thì làm thế nào? Và để làm gì?

- **Định nghĩa:**
- Con trỏ: là **một biến** chứa **địa chỉ bộ nhớ của biến khác**. Nếu x chứa địa chỉ của y thì x được gọi là “**trỏ tới**” y.
- **Cú pháp:** type \*var-name;    type\* var-name;    btype \* var-name;
- **type:** kiểu cơ bản của con trỏ, **var-name:** tên của biến con trỏ

## Ví dụ khai báo

---

- `char *p; // con trỏ kiểu char`
  - `int * p; // con trỏ nguyên, bởi vì kiểu của nó là int.`
  - `float* fp; // con trỏ float, kiểu của nó là float.`
  - `double *dp; //con trỏ, trỏ tới double, bởi vì kiểu của nó là double.`
- **Kiểu cơ bản của một con trỏ:** xác định kiểu dữ liệu của nó trỏ tới – đóng vai trò quan trọng trong thao tác con trỏ.



## 2. Các toán tử con trỏ: (\*, &)

- & toán tử một ngôi, trả về địa chỉ bộ nhớ của toán hạng.

```
int *balptr;
```

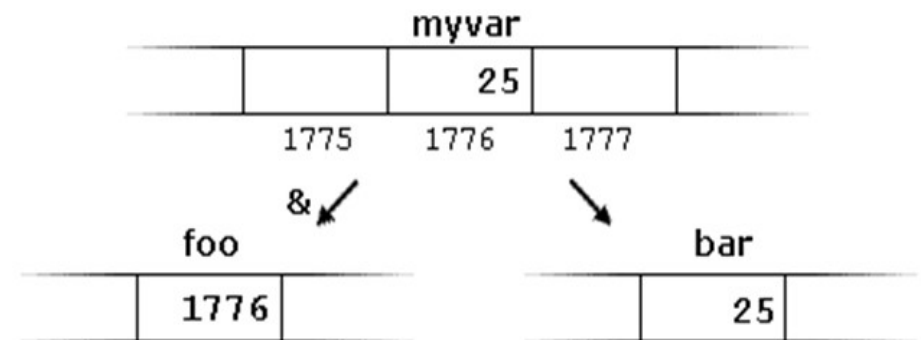
```
balptr = &balance; // & đặt trước biến balance
```

- Tức là: gán địa chỉ bộ nhớ của biến `balance` cho biến `balptr`.
- Ví dụ: Xét đoạn mã sau:

```
myvar = 25;
```

```
foo = &myvar; // chứa địa chỉ
```

```
bar = myvar;
```



- Biến `foo` lưu địa chỉ của biến `myvar` (1776) (gọi là con trỏ).
- Ghi chú:** Địa chỉ của một biến trong bộ nhớ máy tính không thể biết trước khi thời gian chạy. (giả thiết rằng `myvar` đặt ở địa chỉ ô nhớ máy tính là 1776 trong thời gian chạy).



## 2. Các toán tử con trỏ: (\*, &)

- **\*** là toán tử một ngôi, trả về **giá trị** của biến đặt tại địa chỉ cụ thể bởi toán hạng của nó.

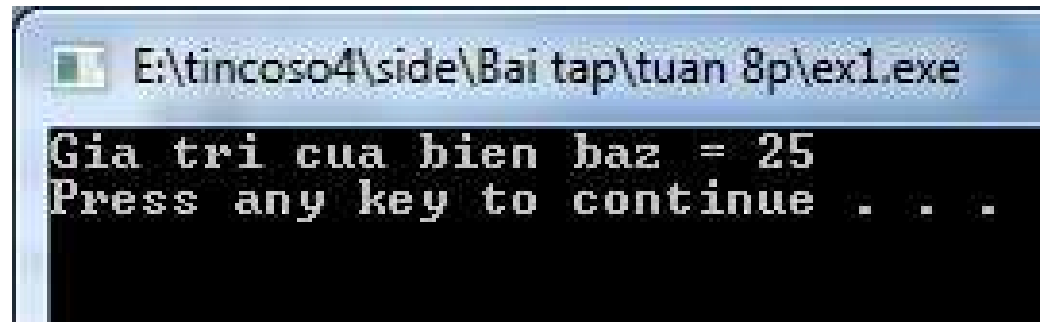
`value = *balptr; // * được đặt trước tên con trỏ.`

- Ví dụ  
`int *foo, myvar = 25;`  
`int baz;`

`foo = &myvar; // foo chứa địa chỉ của biến myvar.`

`baz = *foo; // đọc là: baz bằng giá trị được trỏ bởi foo.`

Tức là: sẽ đặt giá trị của biến **myvar** vào biến **baz**;



The screenshot shows a command prompt window with the title bar "E:\tincoso4\side\Bai tap\tuan 8p\ex1.exe". The window contains the text "Gia tri cua bien baz = 25" and "Press any key to continue . . .".

## Chú ý ban đầu

---

- Sự khác biệt có hoặc không có toán tử (\*).

baz = foo; // baz equal to foo (1776) // int \*bar, \*foo;

baz = \*foo; // baz= giá trị được trả bởi foo (25)

int \*ptr, \* p; // dấu \* để chỉ khai báo con trỏ

// khoảng cách không ảnh hưởng

- Tham chiếu & và toán tử \* có thể được chú giải :
  - + & toán tử địa chỉ đọc là : địa chỉ của ... (&myvar)
  - + \* toán tử tham chiếu đọc là: giá trị được trả bởi.
- Cả hai toán tử & và \* đều có thứ tự ưu tiên cao hơn bất kỳ toán tử toán học nào. (ngoại trừ unary minus)
- Hoạt động sử dụng con trỏ được gọi là gián tiếp (vì truy cập một biến gián tiếp thông qua biến khác)

## Chú ý ban đầu

- Chú ý: dấu (\*) sử dụng khi khai báo con trỏ, chỉ có nghĩa rằng nó là một con trỏ (nó là một phần của kiểu kết hợp). Không được nhầm lẫn với với toán tử lấy giá trị đã được nói ở trên và nó cũng được viết bằng dấu (\*). **Chúng có hai thứ khác nhau biểu diễn cùng một dấu (\*).**

- Ví dụ:

```
int firstvalue = 5;
```

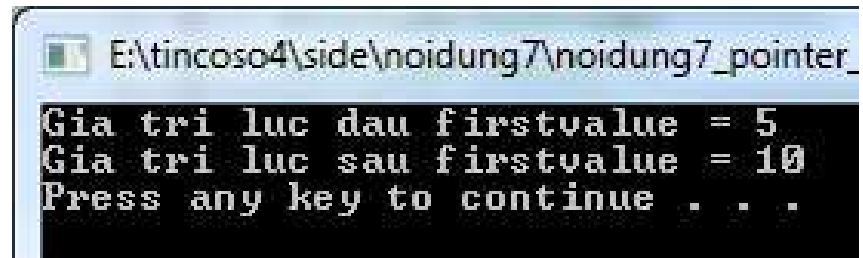
```
cout<<"Gia tri luc dau firstvalue = "<<firstvalue<<endl;
```

```
int * p1;
```

```
p1 = &firstvalue; // p1 = address of firstvalue
```

```
*p1 = 10;          // value pointed to by p1 = 10
```

```
cout<<"Gia tri luc sau firstvalue = "<<firstvalue<<endl;
```



```
E:\tincoso4\side\noidung7\noidung7_pointer_
Gia tri luc dau firstvalue = 5
Gia tri luc sau firstvalue = 10
Press any key to continue . . .
```

## Chú ý ban đầu

- Chú ý:
- Đầu tiên: value1, value2: không được gán giá trị.
- Cuối cùng: cả hai biến được gán một giá trị trực tiếp, thông qua sử dụng con trỏ mypointer.
- **Qui trình**
- Đầu tiên: mypointer được gán địa chỉ của value1 sử dụng toán tử địa chỉ (&)
- Sau đó: giá trị được trỏ bởi con trỏ mypointer được gán giá trị 10.
- Bởi vì tại thời điểm này mypointer đang trỏ tới vị trí ô nhớ của value1. Thực tế là thay đổi giá trị của value1 .

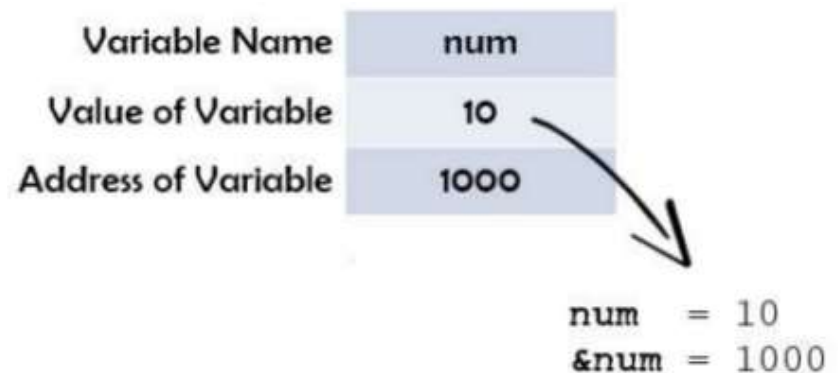
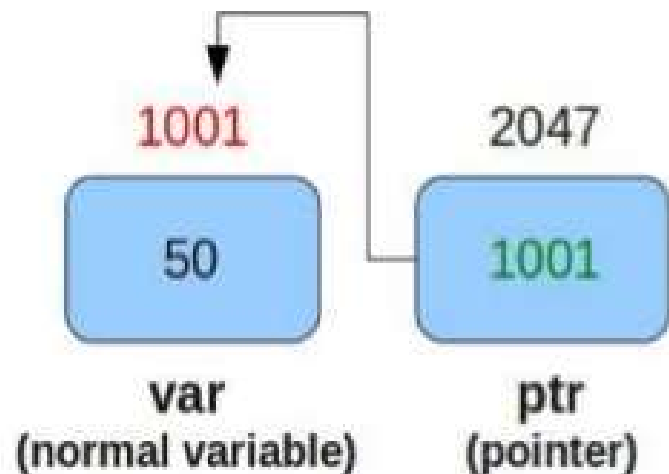
```
int main ()  
{  
    int value1, value2;  
    int * mypointer;  
    mypointer = &value1;  
    *mypointer = 10;  
    mypointer = &value2;  
    *mypointer = 20;  
    cout << "value1 is " << value1 << '\n';  
    cout << "value2 is " << value2 << '\n';  
    return 0; }
```

---

**Output:** value1 is 10, value2 is 20

## Tóm lại

- Biến thông thường: được sử dụng để lưu trữ giá trị.
- Biến con trỏ: được sử dụng để lưu địa chỉ/ tham chiếu đến biến khác.
- Con trỏ là biểu diễn tượng trưng của địa chỉ.
- Chúng ta có thể có một con trỏ trỏ tới bất kỳ kiểu biến nào.



## Tóm lại

---

- Con trỏ có khả năng tham chiếu trực tiếp tới giá trị nó trỏ tới.
- Con trỏ có thuộc tính khác nhau: khi nó trỏ tới char, tới int hoặc float. Một khi truy cập vào vùng nhớ, kiểu con trỏ cần phải biết.
- Khai báo con trỏ cần gộp kiểu dữ liệu mà con trỏ dự định trỏ tới.
- Khai báo một con trỏ theo cú pháp: **type \* name;**

trong đó, type là kiểu dữ liệu được trỏ tới bởi con trỏ. Đây không phải là kiểu của chính con trỏ, mà là kiểu dữ liệu con trỏ trỏ đến.

Ví dụ:

```
int * number;
```

```
char * character;
```

```
double * decimals;
```

## Kiểu cơ bản là quan trọng?

---

- Xét đoạn mã sử dụng con trỏ

```
int balance;  
int *balptr;  
int value;  
balance = 3200;  
balptr = & balance;  
value = *balptr;
```

- **Câu hỏi đặt ra là:** làm cách nào mà C++ biết được bao nhiêu byte để sao chép sang **value** từ địa chỉ được trỏ tới bằng **balptr**?
- Hoặc một cách tổng quát là cách nào mà compiler truyền chính xác số byte cho bất kỳ sự gán nào sử dụng con trỏ?

## Kiểu cơ bản là quan trọng?

---

- **Trả lời:** kiểu cơ bản của con trỏ xác định kiểu dữ liệu mà compiler (biên dịch) giả định con trỏ đang trỏ tới.
- Trong trường hợp này: bởi vì `balptr` là con trỏ nguyên (`int`). C++ copy 4 byte thông tin sang biến **value** từ địa chỉ được trỏ tới bởi `balptr`.
- **NÓI CHUNG:** chúng ta không cần lo lắng về điều này, bởi vì C++ sẽ không cho phép bạn gán một kiểu con trỏ này sang kiểu con trỏ khác trừ khi hai kiểu đó là tương thích.
- Ví dụ: `int *ip;`  
`float *fp;`  
`//.....`  
`ip = &fp; // sai, tuy nhiên nếu ip = (int *) &fp; // thì được`



## Vậy sử dụng ép kiểu có vấn đề gì?

---

- Để hiểu tốt hơn tại sao sử dụng ép kiểu để gán một kiểu con trỏ tới một con trỏ khác không là một ý tưởng tốt.

```
int main() {  
    float x, y, *fp;  
    int *ip;  
    x = 123.23;  
    fp = &x; // 4 byte  
    ip = (int*) &x; // ép kiểu, int chỉ còn 2 byte  
    y = *ip; // what will we do?  
    cout<<"This is using cast float to float: "<<*fp<<endl;  
    cout<<"This is using cast float to int:  "<<y<<endl; // what will  
    this print? Hiện một giá trị rác nào đó.  
    return 0; }
```

### 3. Gán giá trị thông qua con trỏ

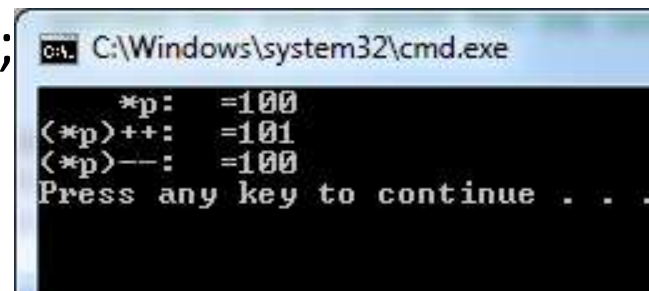
- Nhắc lại thứ tự ưu tiên:

Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(type)	C-style type-casting	

### 3. Gán giá trị thông qua con trỏ

```
int *p, num;  
p = &num; //p chứa địa chỉ của biến num, p trỏ tới num  
*p = 100; // con trỏ bên tay phải của một câu lệnh gán để  
           // gán một giá trị cho vị trí được trỏ bởi con trỏ  
cout<<" *p: ="<<num<<endl;  
(*p)++; // tăng giá trị tại vị trí được trỏ  
cout<<"(*p)++: ="<<num <<endl;  
(*p)--; cout<<"(*p)--: ="<<num << endl;
```

**Output:** 100 101 100



```
C:\Windows\system32\cmd.exe  
*p: =100  
<*p>++: =101  
<*p>--: =100  
Press any key to continue . . .
```

- ❖ Dấu ngoặc () là cần thiết bởi vì toán tử \* có thứ tự ưu tiên thấp hơn ++.

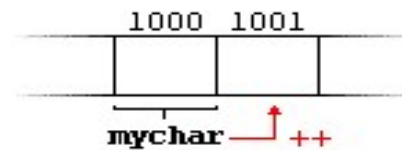
## 4. Số học con trỏ - Pointer arithmetics

---

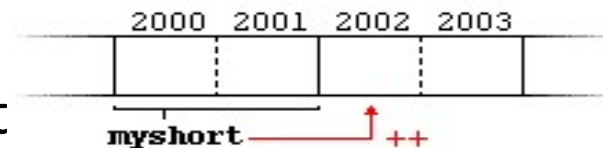
- Để điều khiển các hoạt động số học trên con trỏ là ít khác biệt so với điều khiển số học trên các kiểu số nguyên thông thường.
- Thao tác: cộng và trừ là được phép, còn các phép toán khác là không có ý nghĩa đối với con trỏ.
- Toán tử cộng và toán tử trừ có một chút ứng xử khác biệt với các con trỏ, theo kích cỡ của kiểu dữ liệu nó trỏ tới.
- Kiểu dữ liệu cơ bản: char lấy 1 byte, short lấy 2 byte và int lấy 4 byte....

## 4. Số học con trỏ

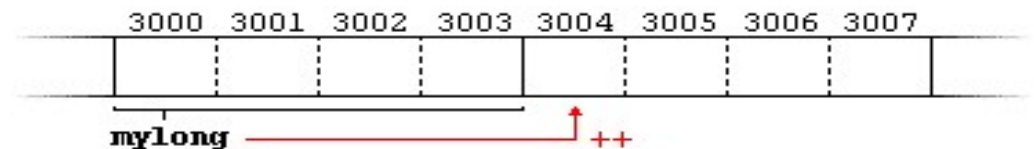
- Có 4 toán tử số học được sử dụng lên các con trỏ: ++, --, +, -
- Ví dụ: `int *p1; // con trỏ nguyên , giả sử chứa địa chỉ 2000.`
- Biểu thức: `++p1; // nội dung của p1 là 2004 không là 2001.`
- Tương tự: `--p1; // nội dung của p1 là 1996 không là 1999.`
- `char *mychar; ++mychar;`



- `short *myshort; ++myshort`



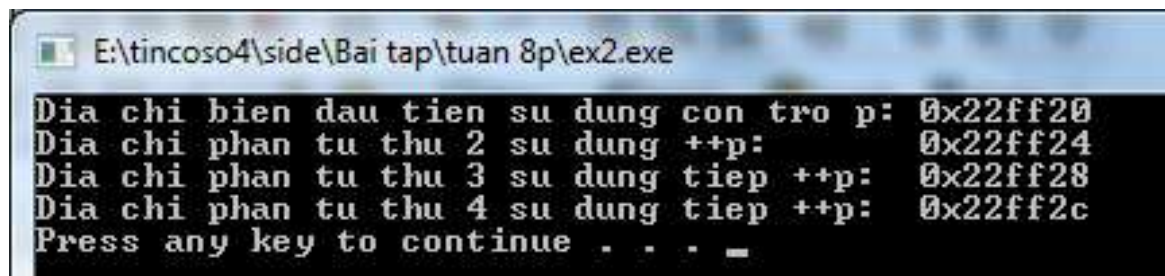
- `long *mylong; ++mylong;`



- Vậy toàn bộ số học con trỏ được thực hiện liên quan đến kiểu cơ bản của con trỏ

## Ví dụ: số học con trỏ

```
int *p;  
int array[4] = {1, 2, 3, 4};  
    // con trỏ vào phần tử đầu tiên  
p = &array[0];  
cout<<"Địa chỉ biến đầu tiên sử dụng con trỏ p: "<<p<<endl;  
cout<<"Địa chỉ phần tử thứ 2 sử dụng ++p:    "<< ++p<<endl;  
cout<<"Địa chỉ phần tử thứ 3 sử dụng tiếp ++p: "<< ++p<<endl;  
cout<<"Địa chỉ phần tử thứ 4 sử dụng tiếp ++p: "<< ++p<<endl;
```

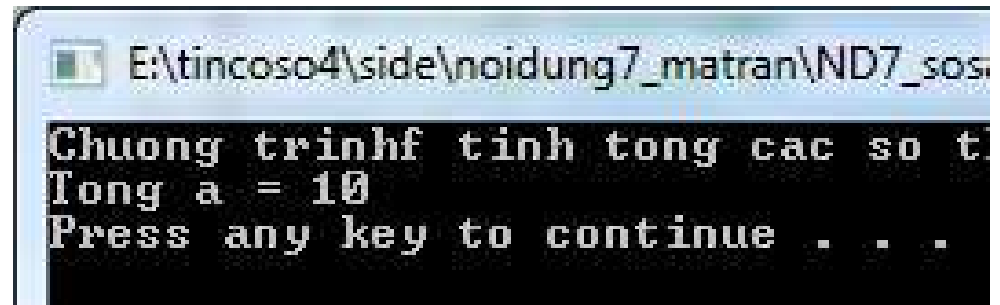


```
E:\tincoso4\side\Bai tap\tuan 8p\ex2.exe  
Địa chỉ biến đầu tiên sử dụng con trỏ p: 0x22ff20  
Địa chỉ phần tử thứ 2 sử dụng ++p: 0x22ff24  
Địa chỉ phần tử thứ 3 sử dụng tiếp ++p: 0x22ff28  
Địa chỉ phần tử thứ 4 sử dụng tiếp ++p: 0x22ff2c  
Press any key to continue . . . _
```

## So sánh con trỏ

- Con trỏ có thể so sánh sử dụng các toán tử quan hệ: `==`, `<`, `>`.
- Kết quả của phép so sánh con trỏ có Ý Nghĩa khi hai con trỏ có mối quan hệ với nhau.
- Ví dụ: nếu `p1`, `p2` là hai con trỏ, trỏ tới 2 biến riêng biệt thì sự so sánh này không có ý nghĩa.
- Tuy nhiên, nếu 2 con trỏ này trỏ tới các biến có quan hệ với nhau (các phần tử của cùng một mảng) thì `p1`, `p2` **có ý nghĩa so sánh**

```
int main() {  
    float a[3]={1, 3, 6};  
    float *p, *pcuoi, s = 0;  
    int n = 3; // so phan tu  
    pcuoi = a + n-1; // dia chi cuoi day  
    for(p = a; p <= pcuoi; ++p) s += *p;  
    cout<<"Tong a = "<<s<<endl;  
} ND7\_sosanh
```



```
E:\tincoso4\side\noidung7_matran\ND7_sosanh  
Chương trìnhf tính tong cac so t  
Tong a = 10  
Press any key to continue . . .
```

## 5. Con trỏ, trỏ tới mảng một chiều

---

- Khái niệm mảng tương đối giống với các con trỏ, các mảng làm việc giống như: **con trỏ trỏ tới phần tử đầu tiên của mảng**.
- Thực tế một mảng luôn có thể được chuyển đổi hoàn toàn sang con trỏ có kiểu thích hợp. Ví dụ :

```
int numbers[5];           //int other_x;  
  
int * p;
```

- Thao tác gán hợp lệ: **p = numbers;** //p = &other\_x
- Sau đó, **p và numbers** sẽ tương đương và có thuộc tính tương tự.
- Trong C/C++ tên mảng là con trỏ và trỏ tới phần tử đầu tiên của mảng.



## 5. Con trỏ trỏ tới mảng một chiều

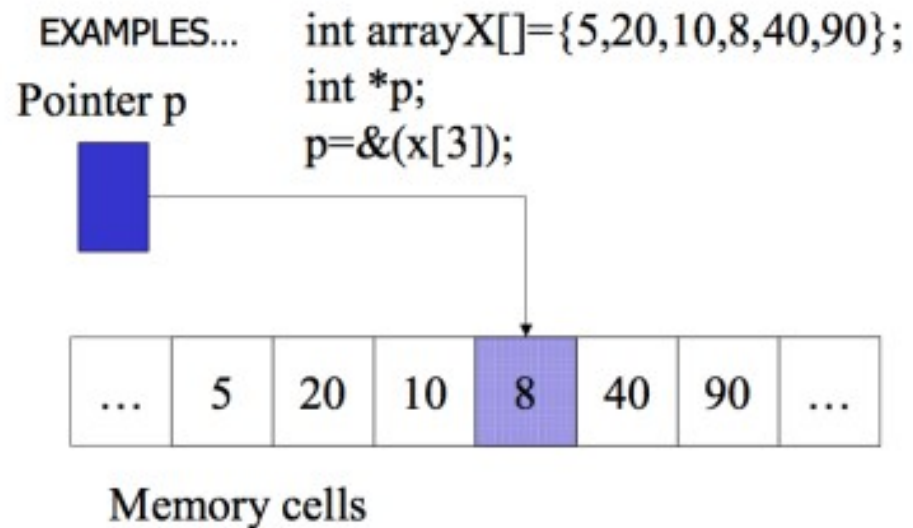
- **Khác biệt chính** là: **p** có thể gán được các địa chỉ khác nhau, trái lại **numbers** không thể và sẽ luôn biểu diễn một khối của 5 phần tử kiểu int. Do vậy, câu lệnh gán sau sẽ không hợp lệ.

`numbers = p;`

- Ví dụ

```
int arrayX[] = {5, 20, 10, 8, 40, 90};
```

```
int *p; p=&x[3];
```



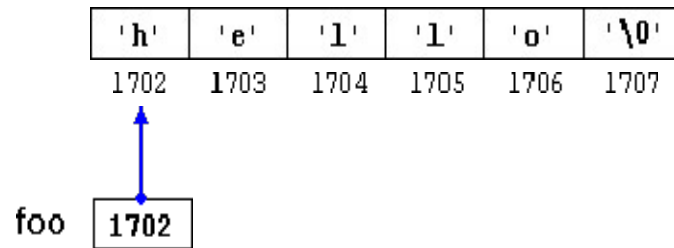
## 5. Con trỏ và mảng

- Trong C++, con trỏ và mảng có mối quan hệ mật thiết với nhau.

- `char foo[80];`

- `char *p1;`

- `p1 = foo;`



- foo là một mảng 80 ký tự, p1 là con trỏ ký tự (char).
- `p1 = foo;` // **p1** được **gán địa chỉ** của **phần tử thứ nhất** trong mảng **foo**, `p1 = &foo, &foo[0]`. Sau câu lệnh gán, p1 sẽ trỏ tới `foo[0]` (tức là chứa địa chỉ của foo).
- Trong C++, sử dụng tên của mảng không cần chỉ số, sinh ra một con trỏ tới phần tử đầu tiên của mảng.
- Sau câu lệnh gán, p1 trỏ tới phần tử đầu tiên của foo, sử dụng p1 để truy cập các phần tử trong mảng.
- `foo[4]` hoặc `*(p1+4)` // trả về phần tử thứ 5,  
// chỉ số mảng bắt đầu từ số 0

## Ví dụ

- [] : thiết lập chỉ số mảng của phần tử.
- Thực tế : [] là Toán tử truy cập vào vùng nhớ mà một con trỏ trỏ tới được gọi là toán tử offset (lệch).
- `a[5] = 0; // a [offset of 5] = 0`  
`*(a+5) = 0; // pointed to by (a+5) = 0`
- tương đương và hợp lệ, không chỉ a là một con trỏ mà a còn là một mảng.

```
int main () {  
    int numbers[5];  
    int * p;  
    p = numbers; *p = 10;  
    p++; *p = 20;  
    p = &numbers[2]; *p = 30;  
    p = numbers + 3; *p = 40;  
    p = numbers; *(p+4) = 50; //p[4]  
    for (int n=0; n<5; n++)  
        cout << numbers[n] << ", ";  
}
```

## 5. Con trỏ và mảng

---

- Chú ý: ngoặc quanh  $*(p1+4)$  là cần thiết vì toán tử  $*$  có ưu tiên lớn hơn toán tử  $+$ . Do đó nếu không có  $()$ : tìm giá trị được trỏ đầu tiên  $+$  với 4  $\Rightarrow$  Chắc chắn đặt đúng dấu vào biểu thức con trỏ.
- Trong C++, có 2 phương pháp truy cập đến phần tử mảng:
  1. Số học con trỏ và
  2. Chỉ số mảng.
- Sử dụng con trỏ để truy cập phần tử mảng là phổ biến trong chương trình C++.
- Ví dụ:

## 5. Con trỏ và mảng

---

- **Lập chỉ số một con trỏ:** Trong C++, có thể lập chỉ số một con trỏ khi nó là một mảng

```
char str[20] = "hello tom";
```

```
char *p;
```

```
int i;
```

```
p = str;
```

```
for(i = 0; p[i]; i++) p[i] = toupper(p[i]);
```

```
cout<<p<<endl;
```

Output: HELLO TOM

- Trong C++, câu lệnh `p[i]` tương đương `*(p+i)`.

## 5. Con trỏ và mảng

---

- Con trỏ và mảng có thể hoán đổi được không?
- Trong nhiều trường hợp, con trỏ và mảng có thể hoán đổi cho nhau. Tuy nhiên, con trỏ và mảng không hoàn toàn hoán đổi.
- `int num[10]; int i;`
- `for (i=0; i<10; i++) {`  
    `*num = i;`  
    `num++; // sai {num++ (num = num + 1)}`  
    `}`
- Sai bởi vì, `num` là **hằng** trỏ tới đầu mảng. Do đó, không thể tăng (hằng là không thể thay đổi).
- `*(num+3) = 100; //` thì hợp lệ - offset – độ lệch

## Mối quan hệ giữa con trỏ và mảng

1. Phép lấy địa chỉ: `double a[20];`

Địa chỉ phần tử thứ  $i$   $[0,19]$  là: `&a[i]`.

2. Tên mảng là một hằng địa chỉ - địa chỉ của phần tử đầu tiên  
(`a++`, `a--` là không được)

`a` tương đương với `&a[0]`;

`a+i` tương đương `&a[i]`

`*(a+i)` tương đương với `a[i]`

3. Hai câu lệnh sau tương đương

`float a[30], *p;`

`p = a;`

Thì bốn câu lệnh sau có tác dụng tương tự

`a[i], *(a+i), *(p+i), p[i]`

## 6. Mảng con trỏ

---

- Giống như bất kỳ kiểu dữ liệu khác.
- `int *pi[10];` // pi là một mảng con trỏ nguyên có kích thước 10  
// (pi – mảng của 10 con trỏ nguyên)
- Để gán địa chỉ của biến nguyên cho địa chỉ của phần tử thứ 3 của mảng con trỏ ta làm như sau:

```
int var;
```

```
pi[2] = &var;
```

- Để tìm giá trị của biến `var` ta làm như sau:  

```
var = *pi[2];
```

- Ví dụ:

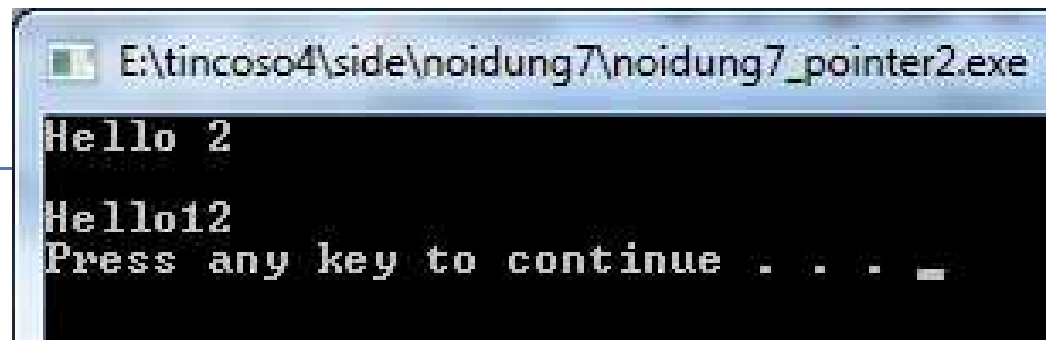


## Ví dụ Mảng con trỏ

int main()

```
{
    char *p1[] = {"Hello 1 \n",
                  "Hello 2 \n",
                  "Hello 3 \n"};
    char *p2[][2] = {"Hello11 ", "Hello12",
                     "Hello21", "Hello22"};
    cout<<p1[1] <<endl; // truy cap
    //p1[1] = "New string"; // thay the
    cout<<p2[0][1]<<endl;

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



```
E:\tincoso4\side\noidung7\noidung7_pointer2.exe
Hello 2
Hello12
Press any key to continue . . . _
```

# Chuyển ký tự từ chữ thường sang chữ hoa và ngược lại

---

```
#include <iostream>  #include <cctype>  using namespace std;
int main() {
    char *p;
    int i;
    char str[80] = "This Is A Test";
    cout << "Chuoi ky tu ban dau: " << str << "\n";
    p = str;
    for(i = 0; p[i]; i++) {
        if(isupper(p[i]))
            p[i] = tolower(p[i]);
        else if(islower(p[i]))
            p[i] = toupper(p[i]);
    }
    cout << "Chuoi sau chuyen doi: " << str;
}
```



```
J:\side\va\pointer.exe
Chuoi ban dau: This Is A Test
Chuoi sau chuyen: tHIS iS a tEST
```

## 7. Khởi tạo con trỏ - Pointer initialization

---

- Con trỏ có thể khởi tạo để trỏ tới một vị trí xác định tại mỗi thời điểm chúng được định nghĩa.

Định nghĩa	Tương đương
<pre>int myvar; int * myptr = &amp;myvar;</pre>	<pre>int * myptr; myptr = &amp;myvar;</pre>

- Con trỏ có thể được khởi tạo ban đầu địa chỉ của biến (như trên) hoặc giá trị của một con trỏ khác (hoặc mảng).

```
int myvar;  
int *foo = &myvar;  
int *bar = foo;
```

- Lưu ý tránh nhầm lẫn: mã sau không hợp lệ

```
int myvar;  
int * myptr;  
*myptr = &myvar; // không phải là giá trị được trỏ
```

## 7. Quy ước con trỏ NULL, nullptr, '\0'

---

- Sau khi một con trỏ được khai báo, nhưng trước khi được gán, nó sẽ chứa một giá trị bất kỳ



- Tránh sử dụng một con trỏ không có khởi đầu. Người lập trình C++ chấp nhận một quy trình tránh một số lỗi. Quy ước.
- Nếu một con trỏ chứa giá trị null (zero). Nó được cho rằng trỏ tới con số không (nothing).**
- Bất kỳ kiểu con trỏ có thể khởi tạo tới null khi nó được khai báo.
- `float *p = '\0'; // nullptr`
- `float *p = NULL ; // dùng trong mã cũ`

## 7. Qui ước con trỏ NULL, nullptr, 0, '\0'

---

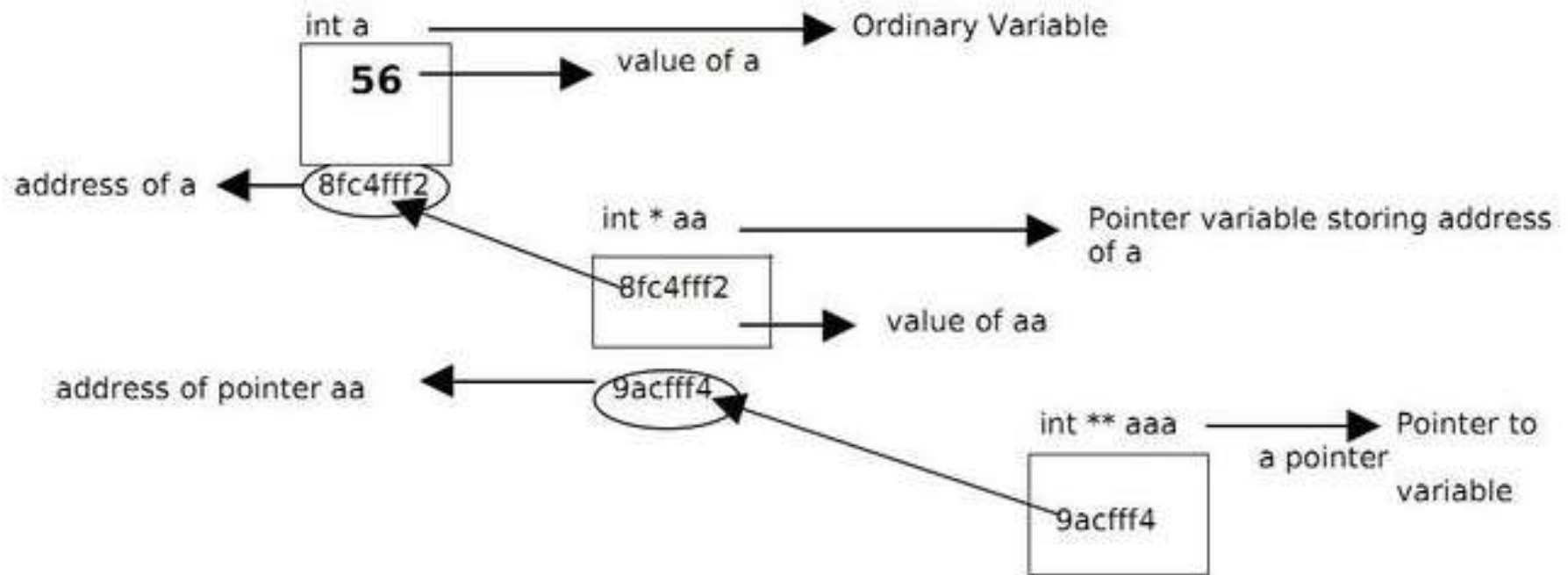
- NULL được định nghĩa trong một vài header của thư viện chuẩn, và được định nghĩa như một bí danh (tên gọi khác) của một vài giá trị hằng con trỏ null (chẳng hạn 0 hoặc với từ khóa nullptr).

```
int * p = 0; //' \0', NULL
```

```
int * q = nullptr; // C++11
```

- Để kiểm tra con trỏ null sử dụng câu lệnh if
- if (p) // nếu p không là null.
- if (!p) // nếu p là null
- Chú ý: tránh nhầm lẫn con trỏ null với con trỏ void. Con trỏ null là một giá trị mà bất kỳ một con trỏ có thể lấy biểu diễn nó là con trỏ tới số 0 (nothing), trong khi con trỏ void là một kiểu của con trỏ có thể trỏ tới một nơi nào đó mà không có kiểu cụ thể.

## 7. Multiple indirection – con trỏ của con trỏ - đa gián tiếp



```
int a, *aa, **aaa;  
a = 56;  
aa = &a;  
aaa = &aa; // **aa = 10;  
cout<<**aaa; // in gia tri cua bien a;
```

[Tạo mảng hai chiều, mảng 2 chiều 2](#)

```
C:\Windows\system32\cmd.exe  
Gia tri cua bien a = 56  
Press any key to continue
```

## Truy cập mảng hai chiều sử dụng con trỏ - truy cập giống mảng 1 chiều

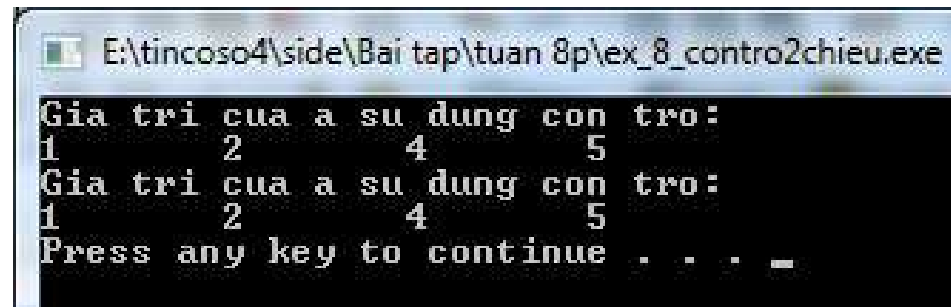
---

```
float a[][2] = {{1,2},
                {4,5}};

float *p;
p = a[0];
for(int i = 0; i<nele;i++) {
    cout <<p[i]<<"\t";
}
```

Hoặc

```
for(int i = 0; i<nele;i++) {
    cout <<*(a[0]+i)<<"\t";
}
```



```
E:\tincoso4\side\Bai tap\tuan 8p\ex_8_contro2chieu.exe
Gia tri cua a su dung con tro:
1 2 4 5
Gia tri cua a su dung con tro:
1 2 4 5
Press any key to continue . . . _
```

## 9. Một số vấn đề với con trỏ

---

### 9.1. Quên không khởi tạo ban đầu cho con trỏ

```
int main() { // ví dụ sai
    int x, *p;
    x = 10;
    *p = x; // p trỏ tới đâu? (thêm p = &x;)
}
```

### 9.2. So sánh không hợp lệ

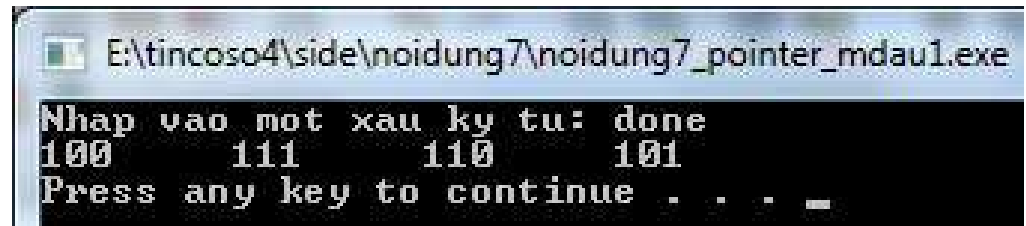
```
char s[80];
char y[80];
char *p1, *p2;
p1 = s;
p2 = y;
if (p1 < p2) // sai
```



## 9. Một số vấn đề với con trỏ

### 9.3. Quên khởi động lại con trỏ

```
int main() {
char s[80];
char *p;
    p = s; // vị trí sai
    do{
//      p = s; // vị trí đúng
        cout<<"Nhập vào một chuỗi ký tự: ";
        gets(s);
        while(*p) {
            cout<<(int) *p++ <<"    "; // in giá trị ASCII
        }
        cout<<"\n";
    }while(strcmp(s,"done"));
} noidung7\_pointer\_mdau1
```



```
E:\tincoso4\side\noidung7\noidung7_pointer_mdau1.exe
Nhập vào một chuỗi ký tự: done
100    111    110    101
Press any key to continue . . . _
```