



IT001 - NHẬP MÔN LẬP TRÌNH

CHƯƠNG 8: CON TRỎ

Sử dụng con trỏ là một trong những kỹ thuật quan trọng trong lập trình, đặc biệt là trong các chương trình được xây dựng bằng ngôn ngữ lập trình C/C++ có liên quan đến việc quản lý bộ nhớ của các biến, cấp phát và thu hồi vùng nhớ trong quá trình thực thi chương trình. Con trỏ hỗ trợ chương trình thực thi nhanh hơn, sử dụng tài nguyên máy tính hiệu quả hơn.

Khoa Khoa học Máy tính



NỘI DUNG

- 8.1 Tổ chức quản lý lưu trữ trong bộ nhớ
- 8.2 Khái niệm con trỏ
- 8.3 Vai trò, tầm quan trọng của con trỏ
- 8.4 Khai báo và khởi tạo biến con trỏ
- 8.5 Các phép toán trên con trỏ
- 8.6 Con trỏ kiểu void
- 8.7 Con trỏ nullptr
- 8.8 Từ khóa const và con trỏ

- 8.9 Con trỏ và mảng một chiều
 - 8.10 Con trỏ và mảng hai chiều
 - 8.11 Cấp phát và giải phóng ô nhớ
 - 8.12 Mảng một chiều cấp phát động
 - 8.13 Con trỏ cấp phát động và chuỗi
 - 8.14 Mảng hai chiều cấp phát động
 - 8.15 Con trỏ và hàm số
- Bài tập



NỘI DUNG

- 8.1 Tổ chức quản lý lưu trữ trong bộ nhớ
- 8.2 Khái niệm con trỏ
- 8.3 Vai trò, tầm quan trọng của con trỏ
- 8.4 Khai báo và khởi tạo biến con trỏ
- 8.5 Các phép toán trên con trỏ
- 8.6 Con trỏ kiểu void
- 8.7 Con trỏ nullptr
- 8.8 Từ khóa const và con trỏ

- 8.9 Con trỏ và mảng một chiều
 - 8.10 Con trỏ và mảng hai chiều
 - 8.11 Cấp phát và giải phóng ô nhớ
 - 8.12 Mảng một chiều cấp phát động
 - 8.13 Con trỏ cấp phát động và chuỗi
 - 8.14 Mảng hai chiều cấp phát động
 - 8.15 Con trỏ và hàm số
- Bài tập



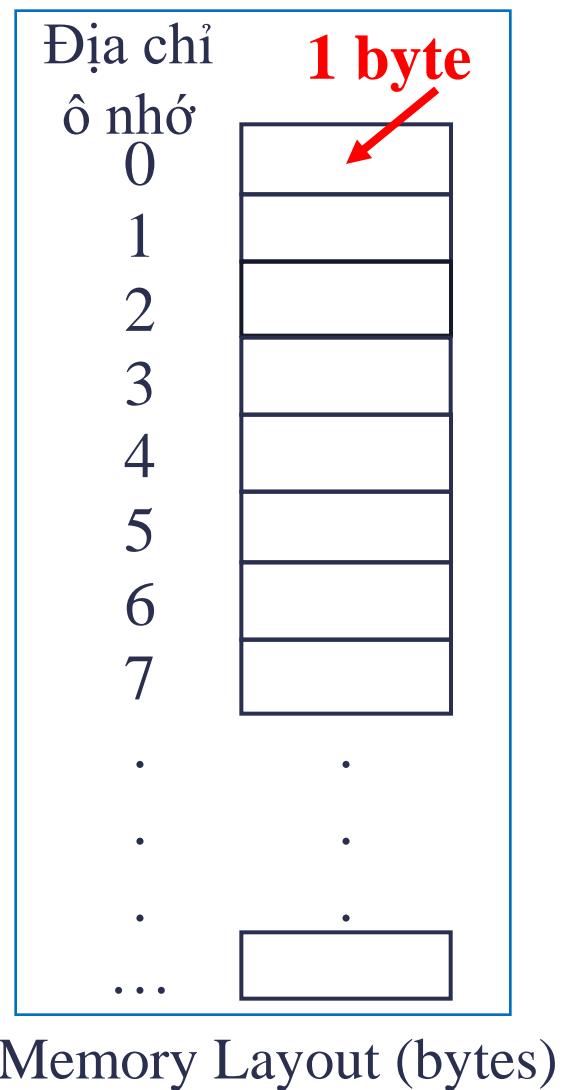
8.1 Tổ chức quản lý lưu trữ trong bộ nhớ



Biến và vùng nhớ

Bộ nhớ máy tính

- Bộ nhớ RAM chứa rất **nhiều ô nhớ**, mỗi ô nhớ có **kích thước 1 byte**.
 - Mỗi ô nhớ **có địa chỉ duy nhất** và địa chỉ này **được đánh số từ 0 trở đi**.
 - RAM để lưu trữ mã **chương trình** và **dữ liệu** trong suốt quá trình thực thi.





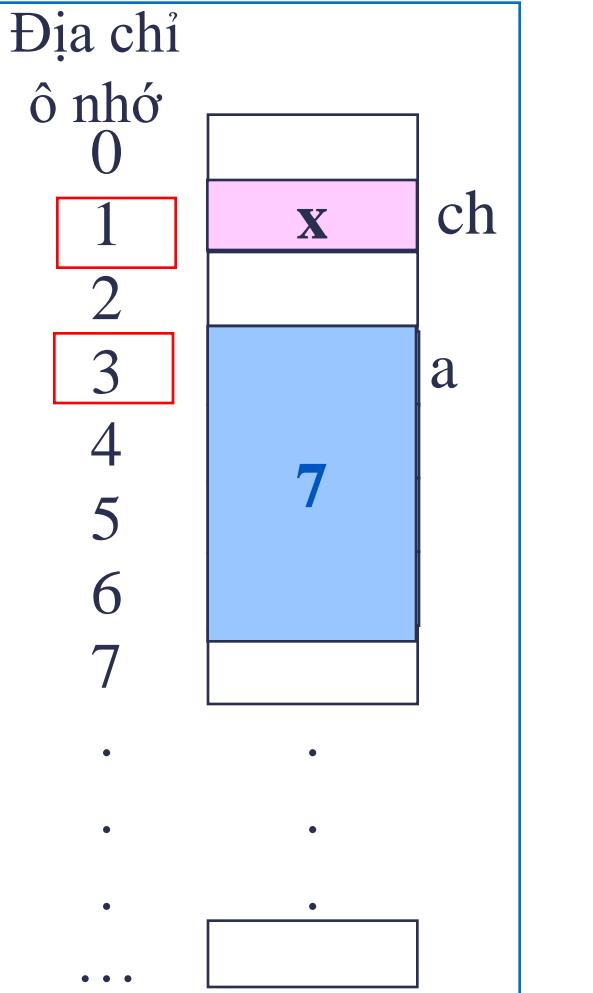
Biến và vùng nhớ

- Khi khai báo biến, máy tính sẽ **dành riêng một vùng nhớ** để lưu biến đó.
- Khi tên biến được gọi, máy tính sẽ thực hiện 2 bước sau:
 - **Tìm kiếm địa chỉ ô nhớ** của biến.
 - **Truy xuất hoặc thiết lập giá trị** của biến được lưu trữ tại ô nhớ đó.



Biến và vùng nhớ

```
int main() {  
    char ch='x';  
    int a = 7;  
}
```





8.2 Khái niệm con trỏ



8.2 Khái niệm con trỏ

- **Khái niệm:**
 - Con trỏ (**Pointer**) là một **biến lưu trữ địa chỉ** của một **địa chỉ bộ nhớ**.
 - Ví dụ: Biến x **chứa địa chỉ** của biến y. Vậy ta nói biến x “trỏ tới” y.
- **Phân loại con trỏ:**
 - Con trỏ kiểu int dùng để chứa địa chỉ của các biến kiểu int. Tương tự ta có con trỏ kiểu **float, double, ...**



8.3 Vai trò, tầm quan trọng của con trỏ



8.3 Vai trò, tầm quan trọng của con trỏ

- **Quản lý bộ nhớ:** Con trỏ cho phép lập trình viên cấp phát và giải phóng bộ nhớ thủ công, tối ưu hóa việc sử dụng bộ nhớ và tránh lãng phí.
- **Truy cập trực tiếp:** Con trỏ cho phép truy cập trực tiếp vào các ô nhớ, giúp thao tác dữ liệu nhanh chóng và hiệu quả.
- **Khả năng linh hoạt:** Con trỏ giúp truy cập các cấu trúc dữ liệu phức tạp như DSLK, cây.
- **Lưu ý:**
 - **Nguy cơ rò rỉ bộ nhớ:** Cần giải phóng bộ nhớ đúng cách để tránh rò rỉ.
 - **Lỗi truy cập vùng nhớ:** Sử dụng con trỏ sai có thể dẫn đến lỗi chương trình.



NỘI DUNG

- 8.1 Tổ chức quản lý lưu trữ trong bộ nhớ
- 8.2 Khái niệm con trỏ
- 8.3 Vai trò, tầm quan trọng của con trỏ
- 8.4 Khai báo và khởi tạo biến con trỏ**
- 8.5 Các phép toán trên con trỏ**
- 8.6 Con trỏ kiểu void
- 8.7 Con trỏ nullptr
- 8.8 Từ khóa const và con trỏ
- 8.9 Con trỏ và mảng một chiều
- 8.10 Con trỏ và mảng hai chiều
- Bài tập



8.4 Khai báo và khởi tạo biến con trỏ



Khai báo con trỏ

- Khai báo

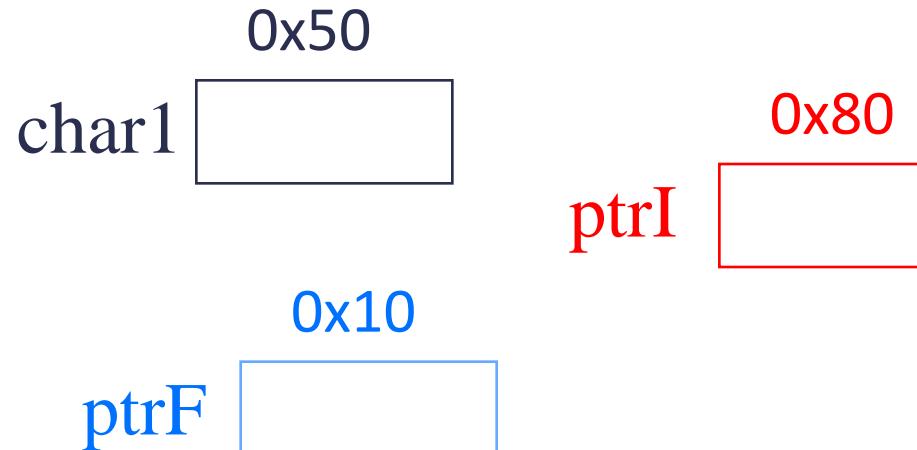
- Giống như mọi biến khác, biến con trỏ muốn sử dụng cũng cần phải được khai báo.

```
<kiểu dữ liệu> *<tên biến con trỏ>;
```

- Ví dụ

```
char char1;  
int *ptrI;  
float *ptrF;
```

Memory Layout





Từ khóa typedef

- Có thể đặt tên cho kiểu dữ liệu con trả dùng **typedef**
 - Ví dụ: **typedef int* IntPtr;**
- ➔ Các khai báo sau tương đương:

IntPtr p;

int *p;



Khởi tạo giá trị con trỏ

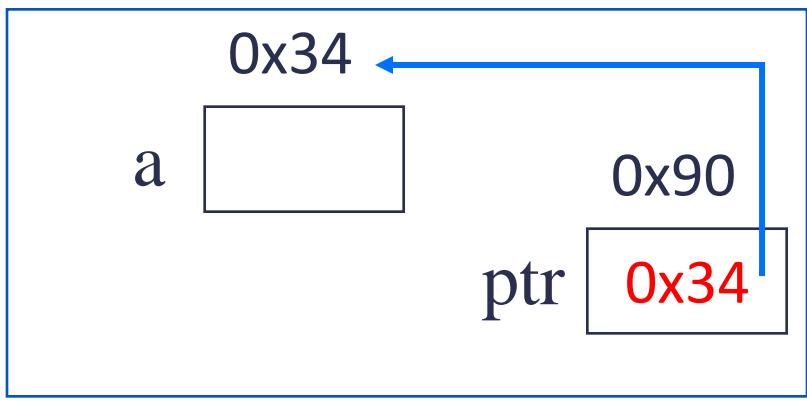
- Toán tử **&** dùng trong khởi tạo giá trị cho con trỏ

```
<kiểu dữ liệu> *<tên biến con trỏ> = & <tên biến>;
```

- Ví dụ:

```
int a;  
int *ptr = &a;
```

? ~~double a;
int *ptr = &a,~~





8.5 Các phép toán trên con trỏ



8.5 Các phép toán trên con trỏ

8.5.1 Con trỏ và toán tử &, *

8.5.2 Toán tử sizeof

8.5.3 Phép gán con trỏ

8.5.4 Phép toán số học trên con trỏ



8.5.1 Con trỏ và toán tử &, *

- Toán tử **& (Address-of Operator)** đặt trước tên biến và cho biết địa chỉ của vùng nhớ của biến.
- Toán tử ***** (**Dereferencing Operator hay Indirection Operator**) đặt trước một địa chỉ và cho biết giá trị lưu trữ tại địa chỉ đó.
- Tương tự, ta sẽ có:
 - Toán tử **&** đặt trước tên biến con trỏ cho biết địa chỉ của vùng nhớ của biến.
 - Toán tử ***** đặt trước biến con trỏ cho phép truy xuất đến giá trị ô nhớ mà con trỏ trỏ đến.



8.5.1 Con trỏ và toán tử &, *

```
int value;  
value = 3200;
```



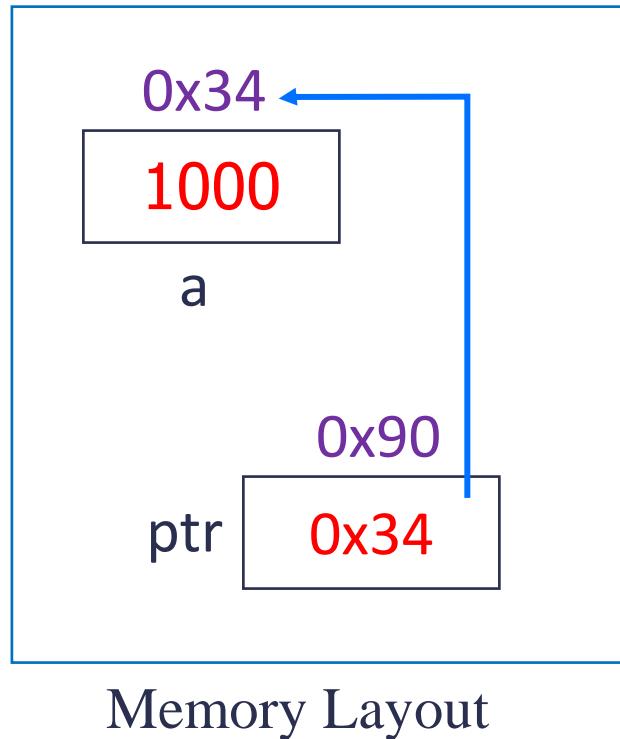
```
cout << " value = " << value;  
=> value = 3200;  
  
cout << " &value = " << &value;  
=> &value = 0x50;  
  
cout << " *(&value) = " << *(&value);  
=> *(&value) = 3200;
```



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
  
cout << &ptr << endl;  
  
cout << ptr << endl  
  
cout << *ptr << endl;  
  
*ptr = 3200;  
  
cout << *ptr;  
  
(*ptr) ++;  
  
cout << *ptr;
```



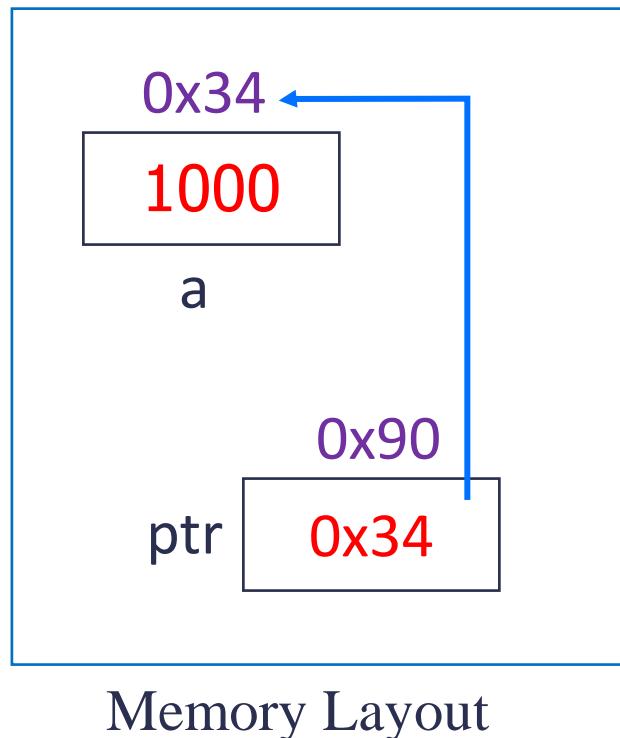
Kết quả thực thi:



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
cout << &ptr << endl;  
cout << ptr << endl  
cout << *ptr << endl;  
  
*ptr = 3200;  
cout << *ptr;  
(*ptr) ++;  
cout << *ptr;
```



Kết quả thực thi:

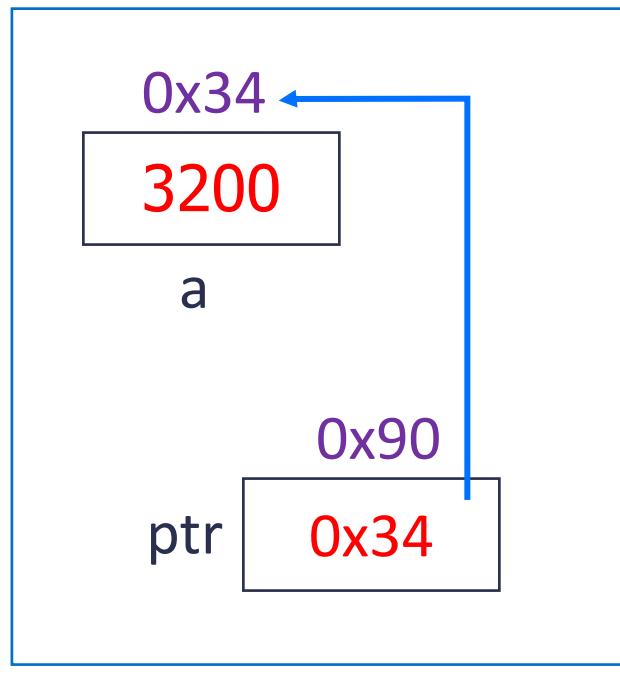
0x90
0x34
1000



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
cout << &ptr << endl;  
cout << ptr << endl  
cout << *ptr << endl;  
*ptr = 3200;  
cout << *ptr;  
(*ptr)++;  
cout << *ptr;
```



Memory Layout

Kết quả thực thi:

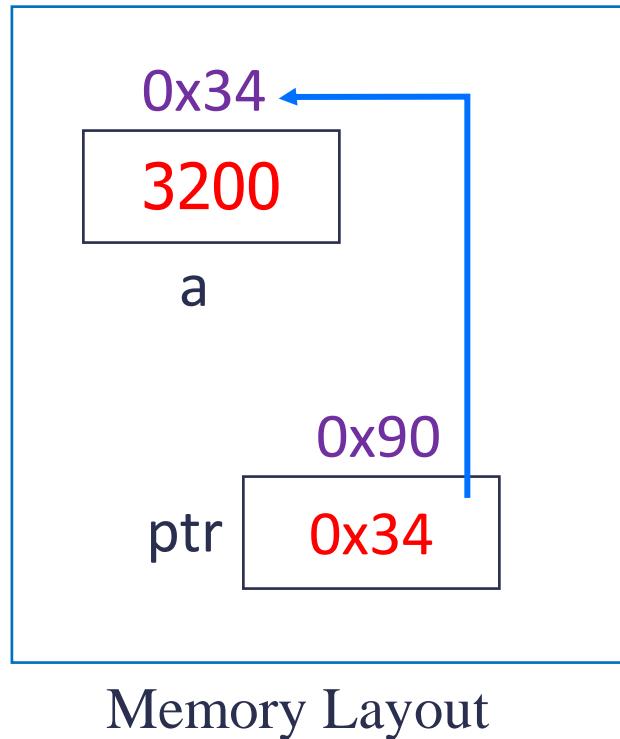
0x90
0x34
1000



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
cout << &ptr << endl;  
cout << ptr << endl  
cout << *ptr << endl;  
*ptr = 3200;  
cout << *ptr;  
(*ptr)++;  
cout << *ptr;
```



Kết quả thực thi:

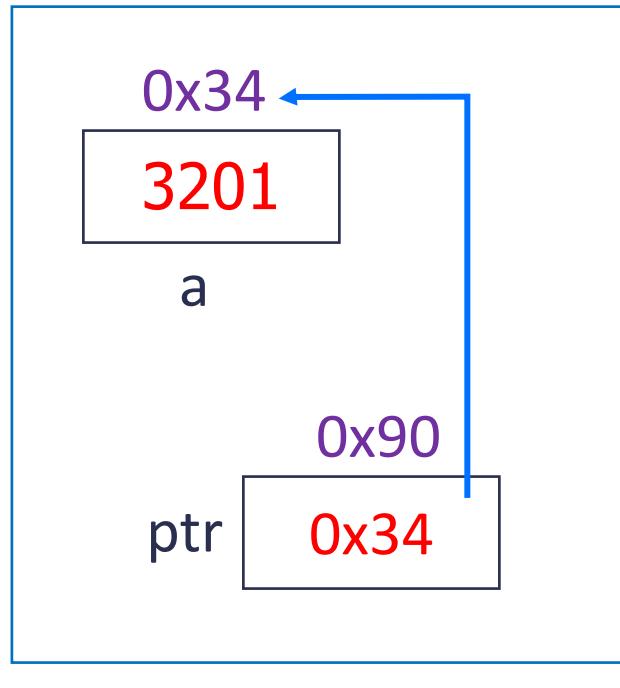
0x90
0x34
1000
3200



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
cout << &ptr << endl;  
cout << ptr << endl  
cout << *ptr << endl;  
*ptr = 3200;  
cout << *ptr;  
(*ptr) ++;  
cout << *ptr;
```



Memory Layout

Kết quả thực thi:

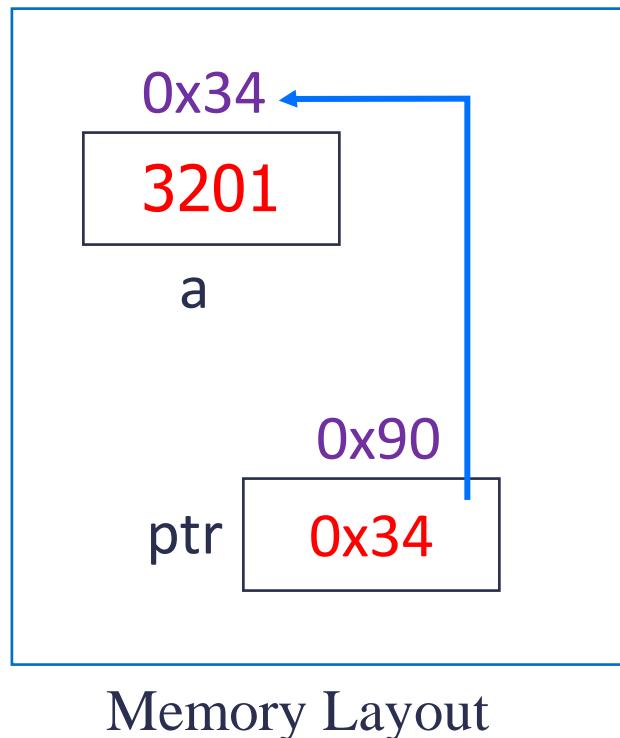
0x90
0x34
1000
3200



8.5.1 Con trỏ và toán tử &, *

- Ví dụ:

```
int a = 1000;  
int *ptr = &a;  
cout << &ptr << endl;  
cout << ptr << endl  
cout << *ptr << endl;  
*ptr = 3200;  
cout << *ptr;  
(*ptr) ++;  
cout << *ptr;
```



Kết quả thực thi:

0x90
0x34
1000
3200
3201



8.5.2 Toán tử sizeof

- Để xác định kích thước (bytes) của một kiểu dữ liệu ta dùng toán tử sizeof.
- Cú pháp: `sizeof (<kiểu_dữ_liệu>)` hoặc `sizeof (<tên_bien>)`
- Con trỏ **chỉ lưu địa chỉ** nên **kích thước của mọi con trỏ là nhau**.
- Kích thước của kiểu dữ liệu con trỏ phụ thuộc vào máy tính và trình biên dịch
- Ví dụ:

<code>int a;</code>	<code>sizeof(a) = 4</code>	<code>sizeof(int) = 4</code>
<code>double b;</code>	<code>sizeof b = 8</code>	<code>sizeof(double) = 8</code>
<code>char c;</code>	<code>sizeof(c) = 1</code>	<code>sizeof(char) = 1</code>
<code>int *pa;</code>	<code>sizeof pa = 8</code>	<code>sizeof(int*) = 8</code>
<code>double *pb;</code>	<code>sizeof pb = 8</code>	<code>sizeof(double*) = 8</code>
<code>char *pc;</code>	<code>sizeof(pc) = 8</code>	<code>sizeof(char*) = 8</code>



8.5.3 Phép gán con trỏ

- Có thể gán biến con trỏ cho con trỏ khác, cũng như có thể gán giá trị cho biến con trỏ đã xác định vùng nhớ trả tới. Ví dụ:

```
int x=10;  
int *p1, *p2=&x;  
p1 = p2;  
*p1=20;
```

→ “Chỉ định p1 trả tới nơi mà p2 đang trả tới”

- Phép gán SAI: gán giá trị đến con trỏ chưa xác định vùng nhớ trả tới.

```
int x=10;  
int *p1, *p2=&x;  
*p1 = 20;
```

→ Gán giá trị x cho “vùng nhớ trả bởi p1”

→ SAI: do p1 chưa xác định vùng nhớ trả tới

```
int x=10;  
int *p1, *p2=&x;  
*p1 = *p2;
```

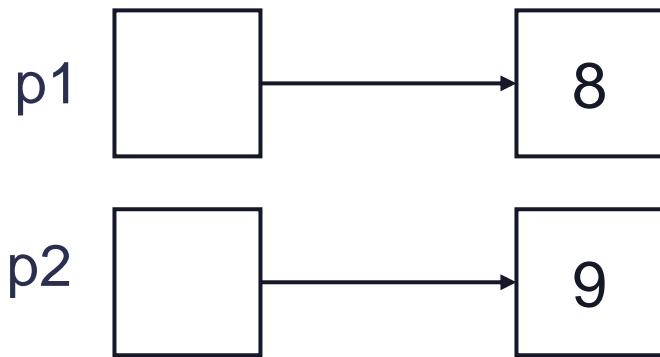
→ Gán “giá trị trả bởi p2” cho “giá trị trả bởi p1”

→ SAI: do p1 chưa xác định vùng nhớ trả tới



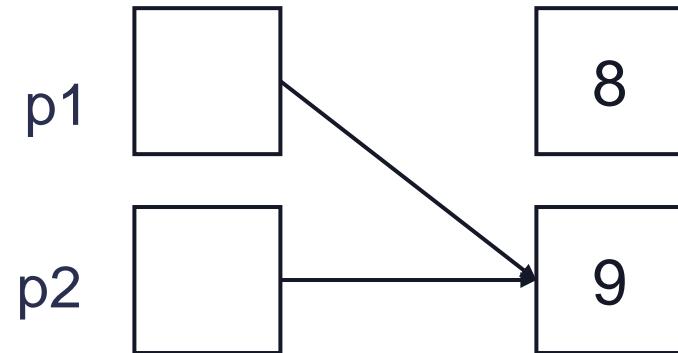
Ví dụ: Phép gán con trỏ

Trước lệnh gán $p1=p2$

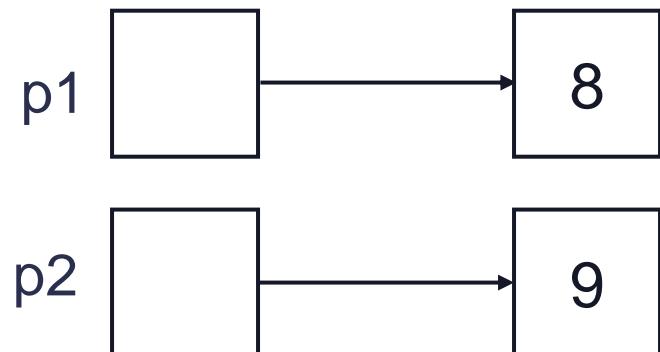


Thực hiện phép gán
 $p1=p2$, ta được:

Sau lệnh gán $p1=p2$

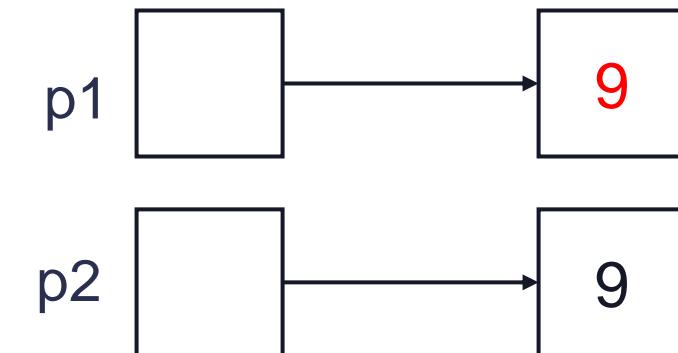


Trước lệnh gán $*p1= *p2$



Thực hiện phép gán
 $*p1= *p2$, ta được:

Sau lệnh gán $*p1= *p2$





8.5.4 Các phép toán số học trên con trỏ

- Để dễ hiểu định nghĩa về các phép toán trên con trỏ, ta cho trước con trỏ p1, p2 có cùng kiểu dữ liệu T và n là một số nguyên. Các phép toán sau là hợp lệ:

- **p1 + n** : Kết quả là con trỏ trỏ đến vị trí cách n phần tử từ p1
- **p1 - n** : Kết quả là con trỏ trỏ đến vị trí cách n phần tử về phía trước của p1
- **++p1** : Di chuyển con trỏ p1 đến phần tử liền kề phía sau (p1 không phải hằng)
- **p1++** : Di chuyển con trỏ p1 đến phần tử liền kề phía sau (p1 không phải hằng)
- **--p1** : Di chuyển con trỏ p1 đến phần tử liền trước đó (p1 không phải hằng)
- **p1--** : Di chuyển con trỏ p1 đến phần tử liền trước đó (p1 không phải hằng)
- **p1 - p2** : Kết quả là khoảng cách (theo số lượng phần tử) giữa hai con trỏ

Trong đó: Phần tử ở đây là 1 vùng nhớ có kích cỡ size(T), con trỏ đang xét trỏ đến các phần tử của cùng một mảng hoặc khối bộ nhớ liên tục.

- Các phép toán so sánh: So sánh địa chỉ giữa hai con trỏ (thứ tự ô nhớ):

``==``, ``!=``, ``>``, ``>=``, ``<``, ``<=``

- Lưu ý: Con trỏ không thể thực hiện các phép toán: ``*``, ``/``, ``%``



8.5.4 Các phép toán số học trên con trỏ

- Xét ví dụ sau: Khai báo con trỏ p1, p2:

```
int *p1, *p2;
```

Giả sử:

- Địa chỉ p1 đang trỏ tới là 0x61fdfc, giá trị đang lưu trữ tại 0x61fdfc là 4
- Địa chỉ p2 đang trỏ tới là 0x61fe08, giá trị đang lưu trữ tại 0x61fe08 là 8
- Ta có:

$$p1 + 1 = 0x61fe00$$

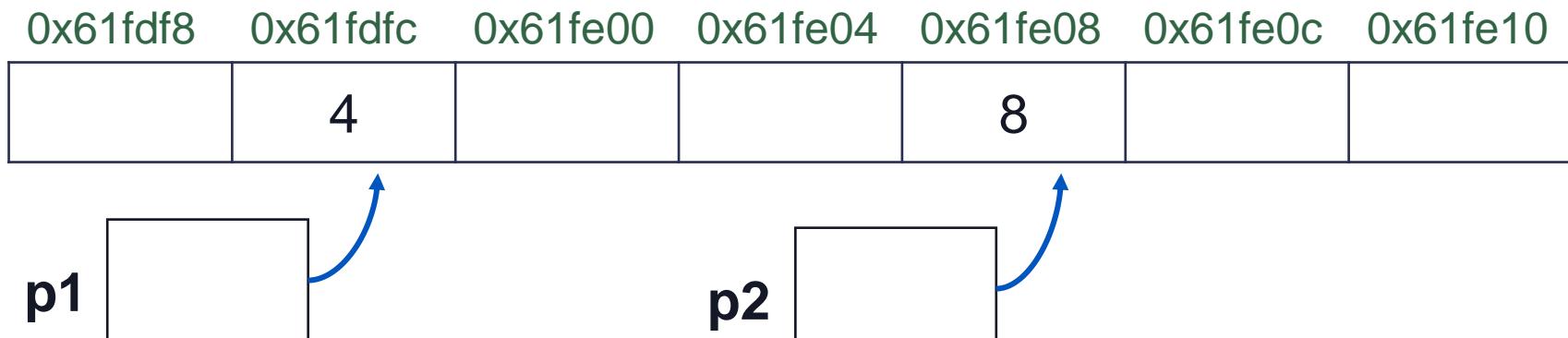
$$p1 - 0 = 0x61fdfc$$

$$p2 - p1 = 3$$

$$p2 + 2 = 0x61fe10$$

$$p2 - 3 = 0x61fdfc$$

$$p1 - p2 = -3$$





NỘI DUNG

- 8.1 Tổ chức quản lý lưu trữ trong bộ nhớ
- 8.2 Khái niệm con trỏ
- 8.3 Vai trò, tầm quan trọng của con trỏ
- 8.4 Khai báo và khởi tạo biến con trỏ
- 8.5 Các phép toán trên con trỏ
- 8.6 Con trỏ kiểu void
- 8.7 Con trỏ nullptr
- 8.8 Từ khóa const và con trỏ
- 8.9 Con trỏ và mảng một chiều
- 8.10 Con trỏ và mảng hai chiều
- Bài tập



8.6 Con trả kiểu void



8.6 Con trả kiểu void

- Con trả void là một kiểu con trả đặc biệt không liên kết với một kiểu dữ liệu cụ thể nào. Con trả void có thể chứa địa chỉ của bất kỳ kiểu dữ liệu nào và có thể được ép kiểu sang bất kỳ kiểu dữ liệu nào.
- Ví dụ:

```
int a = 10;
```

```
char c = 'x';
```

```
void *p = &a;
```

```
p = &c;
```



Ví dụ: Con trả kiểu void

```
#include <iostream>
using namespace std;
int main() {
    int a = 100;
    void *p = &a;
    cout << *p;
    return 0;
}
```

➔ Compiler Error: 'void*' is not a pointer-to-object type

```
#include <iostream>
using namespace std;
int main() {
    int a = 100;
    void *p = &a;
    cout << *(int*)p;
    return 0;
}
```

➔ Đúng



8.7 Con trả nullptr (C++11)



Con trỏ NULL

- **NULL** là một hằng số có giá trị bằng 0 (zero), được định nghĩa dưới dạng một macro trong các thư viện: `<cstddef>` `<cstdlib>` `<cstring>` `<cwchar>` `<ctime>` `<locale>` `<stdio>`: `#define NULL 0`
- Một biến con trỏ có giá trị là **NULL** thường được gọi là con trỏ **NULL**.
- Con trỏ có giá trị **NULL** có nghĩa là con trỏ không trỏ đến bất kỳ đối tượng nào.
- Ví dụ: khai báo “`int *p=NULL;`” tương đương với “`int *p=0`”.
- Lưu ý:

```
int *p=NULL;  
cout << *p;
```

➔ Lỗi “Cannot access memory at address 0x0”



Con trỏ nullptr (C++11)

- `nullptr` là một từ khóa mới được giới thiệu trong C++11. Nó đại diện cho một kiểu mới có tên là `std::nullptr_t`.
- `nullptr` cũng được sử dụng để biểu thị rằng một con trỏ không trỏ đến bất kỳ đối tượng nào.
- `nullptr` được sử dụng từ **C++11** thay cho `NULL` (`NULL` vẫn dùng được trong C++11 nhưng không an toàn)
 - `nullptr` có kiểu dữ liệu riêng là `std::nullptr_t`, giúp phân biệt rõ ràng với các giá trị null khác như 0 hoặc `NULL`.
 - `nullptr` được hỗ trợ bởi tất cả các trình biên dịch C++ hiện đại, giúp đảm bảo tính tương thích cho các chương trình C++.



Con trỏ nullptr (C++11)

- Ví dụ: Hỏi 4 đoạn code bên dưới đoạn nào **không an toàn hoặc không hợp lệ**?

```
// Đoạn 1:  
int *p1 = nullptr;  
if (p1 != nullptr)  
    cout << (*p1)++;  
else cout << "p1 null";
```

```
// Đoạn 2:  
int *p2;  
if (p2 != nullptr)  
    cout << (*p2)++;  
else cout << "p2 null";
```

```
// Đoạn 3:  
int n;  
int *p3 = &n;  
if (p3 != nullptr)  
    cout << (*p3)++;  
else cout << "p3 null";
```

```
// Đoạn 4:  
int* p4 = nullptr;  
*p4 = 10;
```



8.8 Tùy khóa const và con trỏ



8.8 Từ khóa const và con trỏ

- Hằng số dùng trong khai báo một biến cho biết giá trị của biến không được phép thay đổi trong quá trình thực hiện chương trình.
- Tùy thuộc vào vị trí đặt từ khóa **const** dùng trong khai báo biến con trỏ, mà quy định giá trị hằng cho con trỏ hay cho vùng nhớ con trỏ trỏ tới.
- Có 3 trường hợp trong khai báo biến con trỏ và từ khóa **const**.



8.8 Từ khóa const và con trỏ

- Ta xét đoạn lệnh sau:

```
1. int x;  
2. int * p1 = &x;  
3. const int * p2 = &x; // TH1: non-const pointer to const int  
4. int * const p3 = &x; // TH2: const pointer to non-const int  
5. const int * const p4 = &x; // TH3: const pointer to const int
```

- Xét Dòng lệnh 2: `int * p1 = &x;` → đây là phép khởi tạo con trỏ: p1 và x có thể thay đổi giá trị sau đó.
- Ví dụ:

```
int x=20, y=10;  
int* p=&x;  
*p=30; → CHO PHÉP  
p=&y; → CHO PHÉP
```



Trường hợp 1: Con trỏ tới một giá trị hằng

- **Cú pháp:**

```
<Kiểu_Dữ_Liệu_Y> <Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;  
const <Kiểu_Dữ_Liệu_Y> * <Tên_Biến_Con_Trỏ> = &<Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;
```

- **Ý nghĩa:** không thể thay đổi giá trị tại vùng nhớ mà con trỏ đang trỏ tới, nhưng con trỏ có thể thay đổi trỏ tới địa chỉ khác.

- **Ví dụ:**

```
int x=20, y=10;  
  
const int* p=&x; // non-const pointer to const int  
  
*p=30; → LỖI:  
  
p=&y; → CHO PHÉP
```



Trường hợp 2: Con trỏ hằng tới một giá trị không hằng

- **Cú pháp:**

```
<Kiểu_Dữ_Liệu_Y> <Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;  
<Kiểu_Dữ_Liệu_Y> * const <Tên_Biến_Con_Trỏ> = &<Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;
```

- **Ý nghĩa:** Con trỏ không được phép thay đổi địa chỉ đang trỏ tới (read-only), nhưng vùng nhớ con trỏ trỏ tới có thể thay đổi giá trị.

- **Ví dụ:**

```
int x=20, y=10;  
  
int* const p=&x; // const pointer to non-const int  
  
*p=30; → CHO PHÉP  
  
p=&y; → LỖI
```



Trường hợp 3: Con trỏ hằng tới một giá trị hằng

- **Cú pháp:**

```
<Kiểu_Dữ_Liệu_Y> <Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;
```

```
const <Kiểu_Dữ_Liệu_Y> * const <Tên_Biến_Con_Trỏ> = &<Tên_Biến_Có_Kiểu_Dữ_Liệu_Y>;
```

- **Ý nghĩa:** Con trỏ và cả vùng nhớ con trỏ đang trỏ tới không được phép thay đổi giá trị (read-only).

- **Ví dụ:**

```
int x=20, y=10;
```

```
const int* const p=&x; // const pointer to const int
```

```
*p=30; → LỖI
```

```
p=&y; → LỖI
```



8.8 Từ khóa const và con trỏ

- Ví dụ: Tìm lỗi sai trong đoạn code sau:

```
int a = 12;  
int a1 = 9;  
int * const pa = &a;  
pa = &a1;  
const int * pb = &a;  
*pb = a1;  
const int * const pc = &a;  
(*pc) = a1;
```



8.9 Con trỏ và mảng một chiều



8.9 Con trỏ và mảng một chiều

8.9.1 Mảng 1 chiều và cách lấy địa chỉ

8.9.2 Mảng 1 chiều và hằng con trỏ

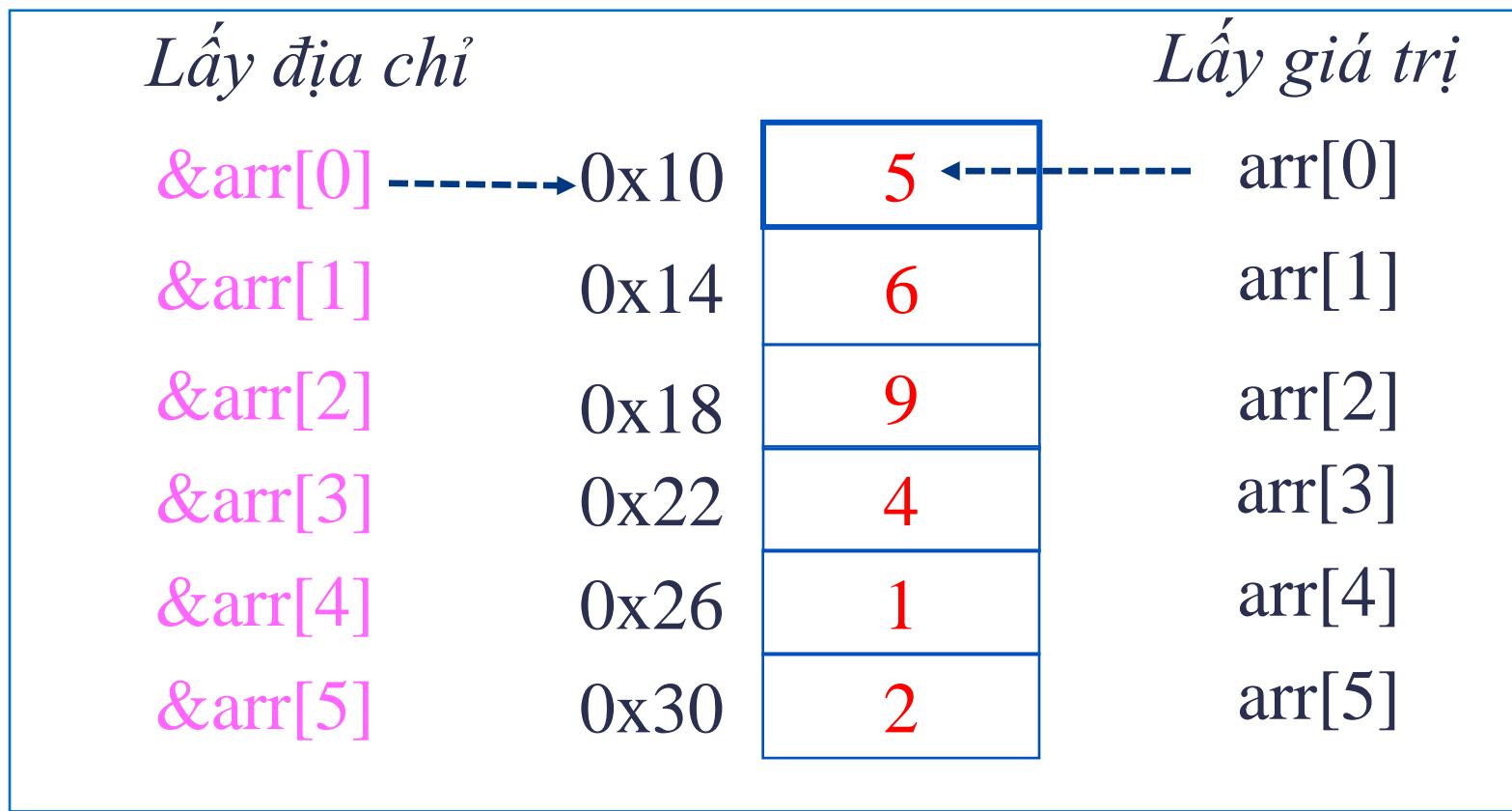
8.9.3 Truy xuất mảng 1 chiều sử dụng con trỏ

8.9.4 Các phép toán số học của con trỏ trên mảng



8.9.1 Mảng 1 chiều và cách lấy địa chỉ

- Cho mảng 1 chiều: `int arr[6] = {5, 6, 9, 4, 1, 2};`





8.9.2 Mảng 1 chiều và hằng con trỏ

- Cho mảng 1 chiều:

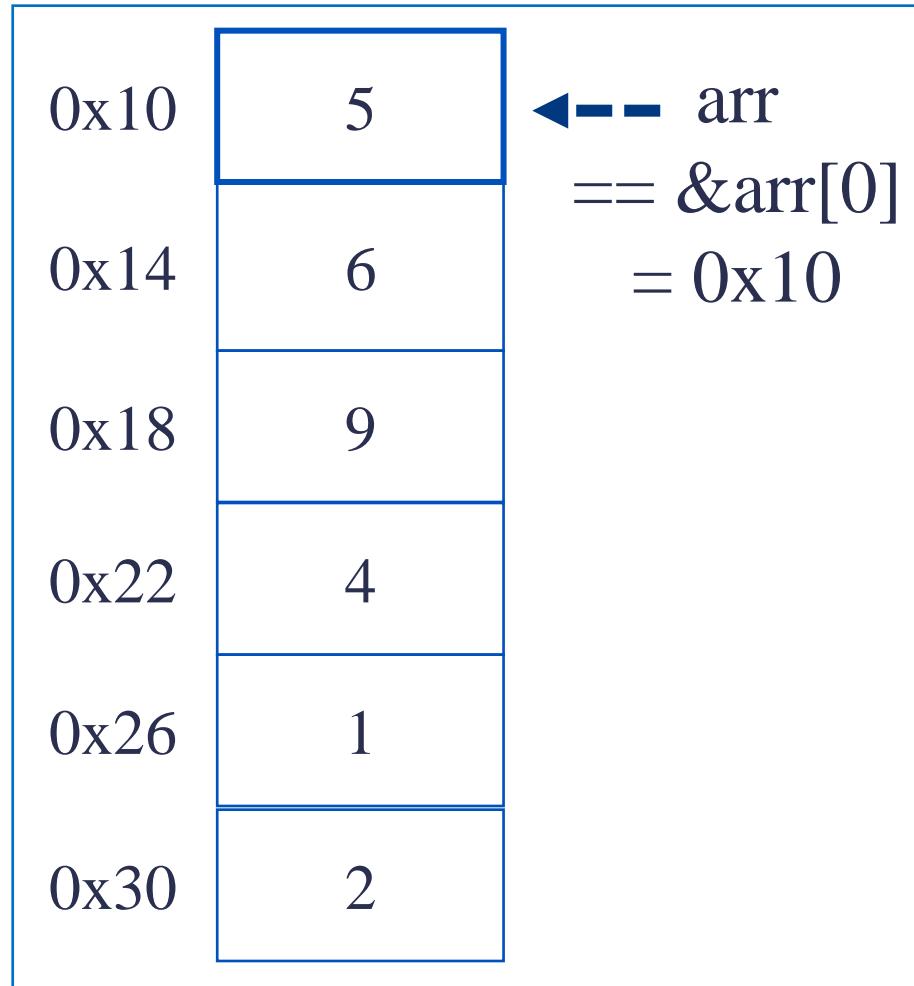
```
int arr[6] = {5, 6, 9, 4, 1, 2};
```

Tên mảng arr là một **hằng con trỏ**

→ không thể thay đổi giá trị của hằng này.

arr là địa chỉ đầu tiên của mảng

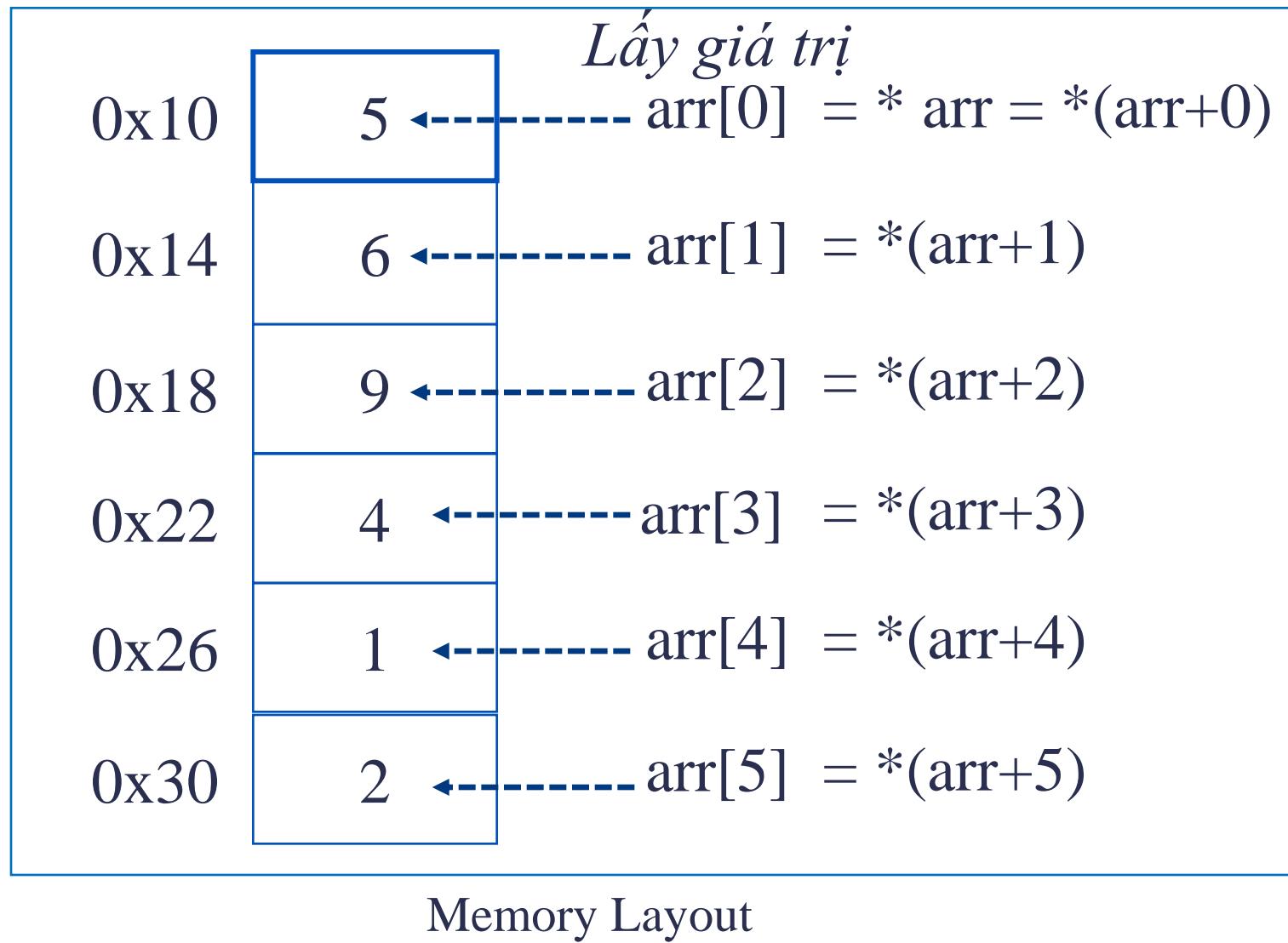
→ $\text{arr} == \&\text{arr}[0]$



Memory Layout



8.9.2 Mảng 1 chiều và hằng con trỏ





Câu hỏi

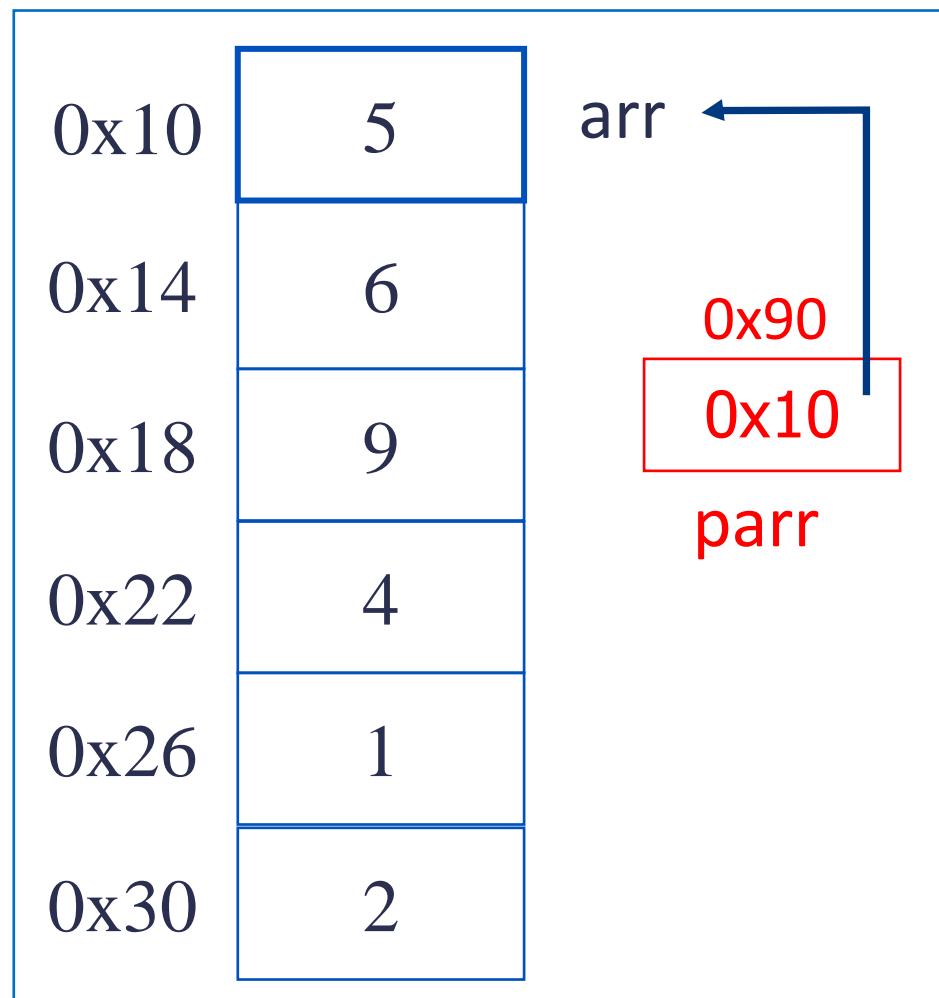
- Tìm lỗi trong đoạn chương trình sau:

```
int num[10];  
  
for (int i = 0; i<10; i++) {  
  
    // num[i]=i;  
  
    // *(num+i)=i;  
  
    *num = i;  
  
    num++;  
  
}  
  
*(num + 3) = 100;
```



8.9.3 Truy xuất mảng 1 chiều sử dụng con trỏ

```
int arr[6]={5, 6, 9, 4, 1, 2};  
  
int *parr;  
  
// Cách 1  
  
parr = arr;  
  
// Cách 2  
  
parr = &arr[0];
```





8.9.3 Truy xuất mảng 1 chiều sử dụng con trỏ

- Truy xuất tới **giá trị** của phần tử thứ *i* của mảng (xét *i* là chỉ số hợp lệ của mảng):

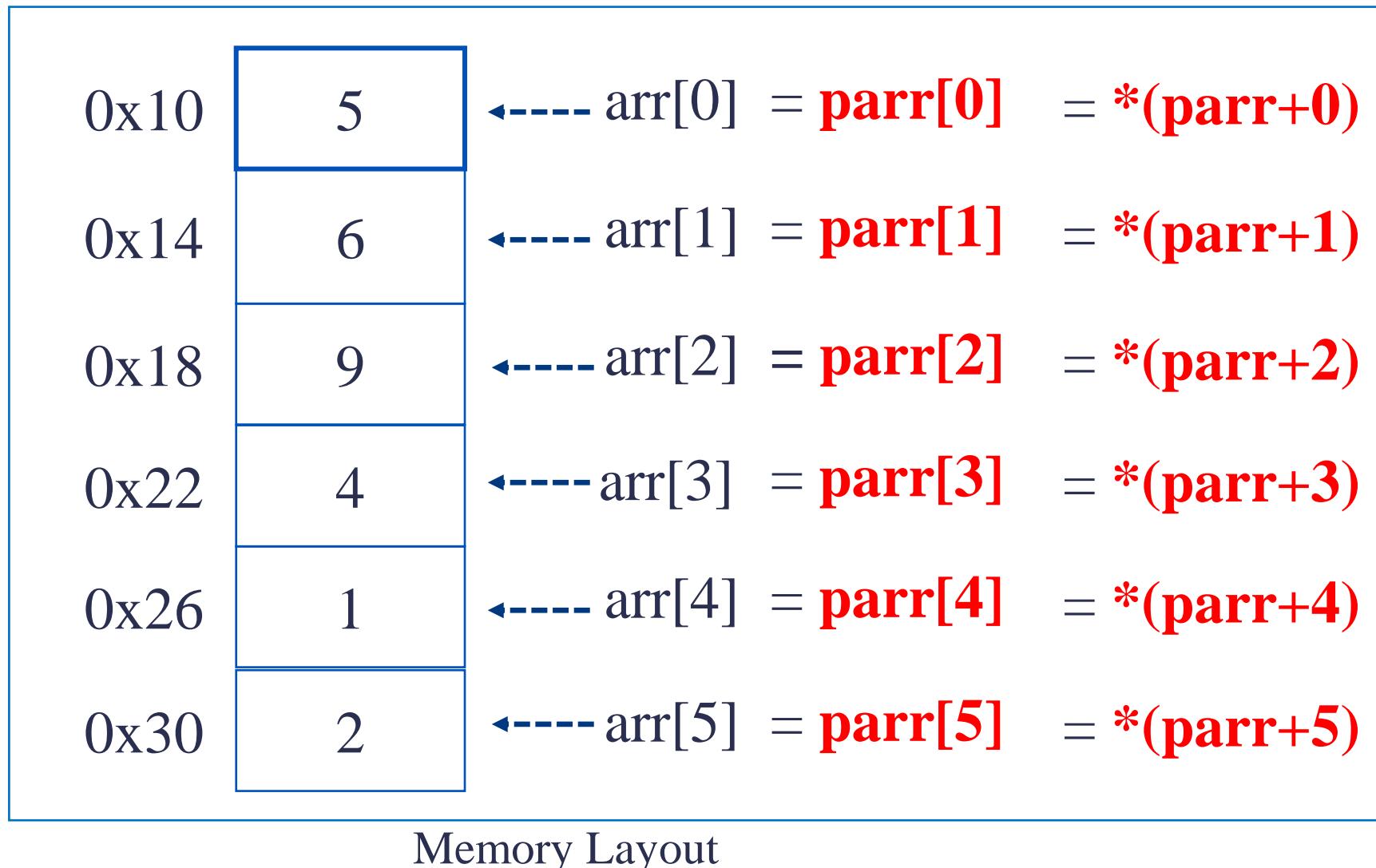
arr[i] == *(arr+i) == parr[i] == *(parr + i)

- Truy xuất tới **địa chỉ** của phần tử thứ *i* của mảng (xét *i* là chỉ số hợp lệ của mảng):

&arr[i] == arr+i == &parr[i] == parr + i

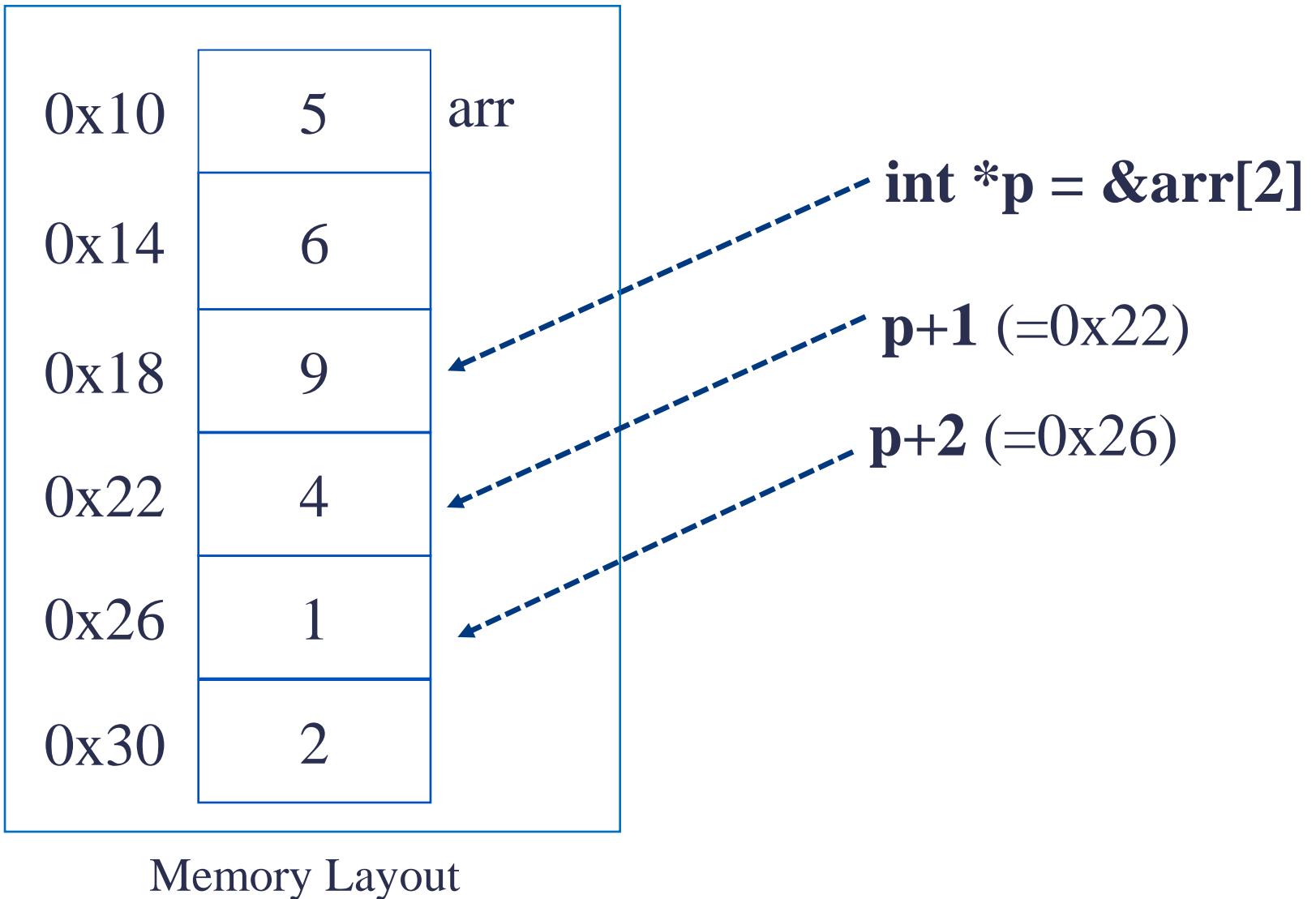


8.9.3 Truy xuất mảng 1 chiều sử dụng con trỏ



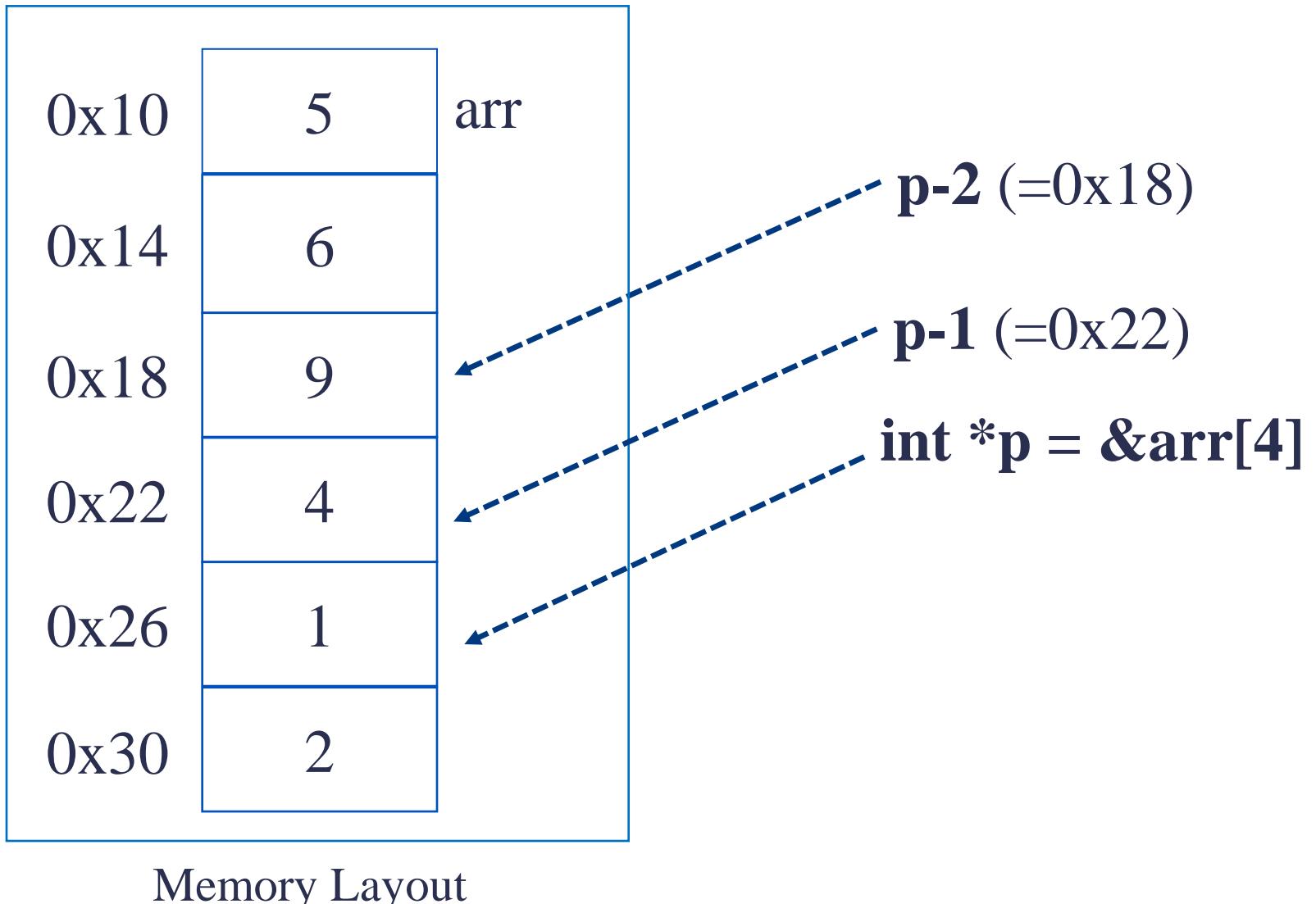


8.9.4 Các phép toán số học của con trỏ trên mảng



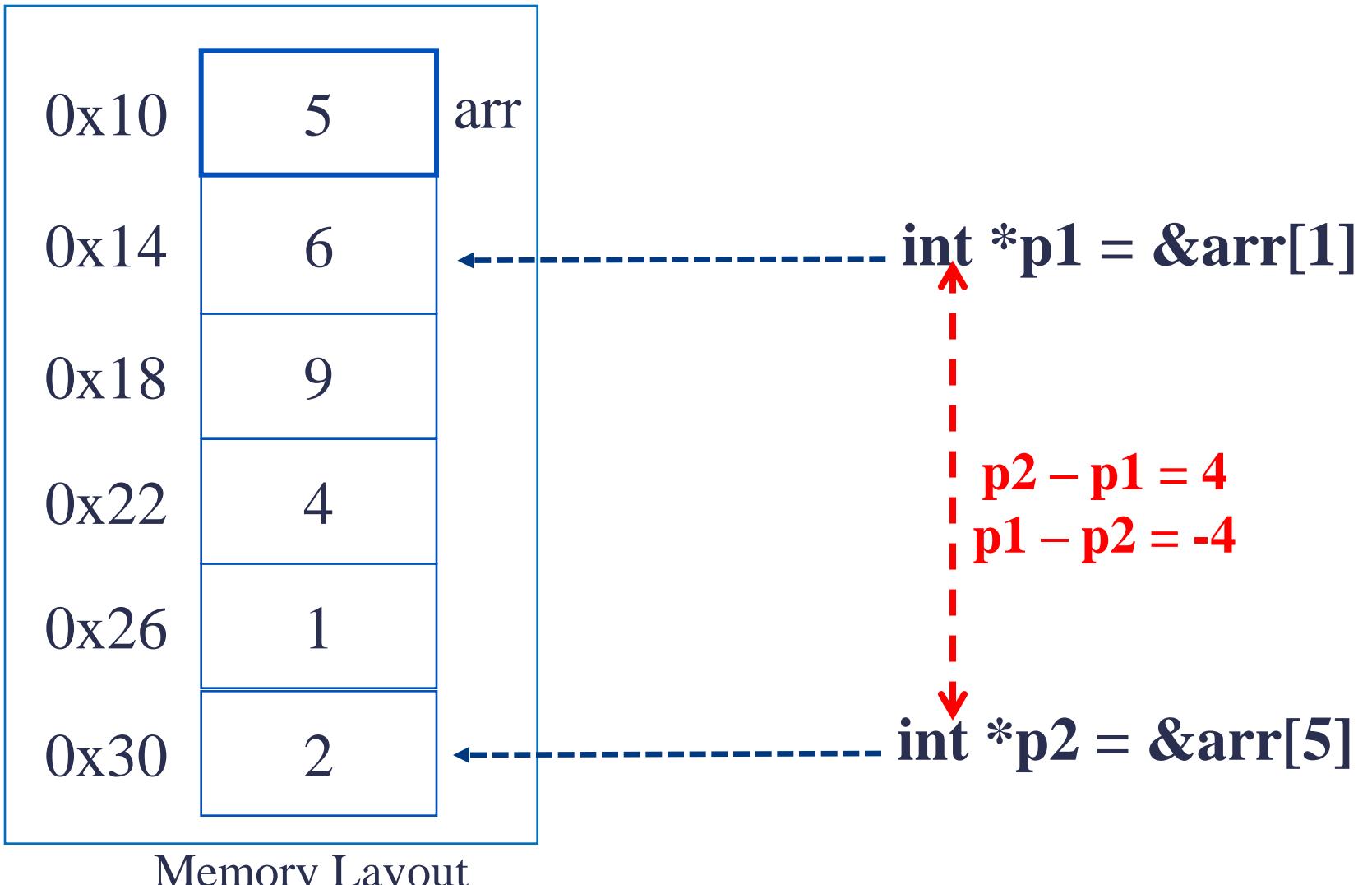


8.9.4 Các phép toán số học của con trỏ trên mảng





8.9.4 Các phép toán số học của con trỏ trên mảng





Bài tập 2

- Cho mảng 1 chiều a có 10 phần tử, biến con trỏ p trỏ tới mảng 1 chiều a.
 - a. Hãy dùng con trỏ p để gán giá trị 100 cho phần tử thứ 5 của mảng.
 - b. Hãy viết chương trình nhập và xuất mảng 1 chiều thông qua con trỏ p.



Lời giải

```
#include <iostream>
using namespace std;
const int n = 10;
int main() {
    int a[n], *p = a;
    *(p+5) = 100; // câu a
    for (int i = 0; i < n; i++) // câu b
        cin >> *(p + i);
    for (int i = 0; i < n; i++) // câu b
        cout << *(p + i) << " ";
}
```



Bài tập 3

- Hãy viết chương trình tạo mảng 1 chiều có n phần tử bằng 2 cách:
 - Cách 1: Bằng mảng chuẩn
 - Cách 2: Bằng cấp phát động



8.10 Con trỏ và mảng hai chiều



8.10 Con trỏ và mảng hai chiều

- Chạy đoạn code sau:

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6}};

int r=2, c=3;

for(int i=0; i<r; i++){
    for(int j=0; j<c; j++)
        cout << &a[i][j] << " ";
    cout << endl;
}

for(int i=0; i<r; i++)
    cout << a[i] << " ";
```

Kết quả thực thi:

0x61fdf0	0x61fdf4	0x61fdf8
0x61fdfe	0x61fe00	0x61fe04
0x61fdf0	0x61fdfe	

Hỏi kết quả của các lệnh tiếp theo sau chương trình bên?

```
cout << a[0] << endl;
cout << a[0]+1 << endl;
cout << a[0]+2 << endl;
cout << a[0]+3 << endl;
cout << a[0]+4 << endl;
cout << a[0]+5 << endl;
```



8.10 Con trỏ và mảng hai chiều

- Xét mảng 2 chiều sau:

```
T a[MAXR][MAXC];
```

- Trong đó: T là kiểu dữ liệu
- Ta có các cách sau để khai báo con trỏ quản lý mảng 2 chiều a:

```
T *p= (T*)a; // Ép kiểu mảng a thành con trỏ cấp 1
```

```
T (*p1)[MAXC]= a; // p1 là con trỏ đến mảng 2 chiều
```

...



8.10 Con trỏ và mảng hai chiều

- Ví dụ 1:

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6}};

int r=2, c=3;

int (*p)[3] = a;

for(int i=0; i<r; i++)
    for(int j=0; j<c; j++)
        cout << p[i][j] << " ";
```

- Ví dụ 2:

```
int a[2][3] = { {1, 2, 3}, {4, 5, 6}};

int r=2, c=3;

int *p = (int*)a;

for(int i=0; i<r*c; i++) cout << p[i] << " ";
```

Kết quả thực thi
của cả 2 ví dụ:

1 2 3 4 5 6



BÀI TẬP



Bài tập

- Câu 1: Toán tử nào dùng để xác định địa chỉ của một biến?
- Câu 2: Toán tử nào dùng để xác định giá trị của biến do con trỏ trỏ đến?
- Câu 3: Phép lấy giá trị gián tiếp là gì?
- Câu 4: Cho biến daa kiểu int. Khai báo và khởi tạo con trỏ pdaa trỏ đến biến này. Sau đó gán giá trị 100 cho biến daa sử dụng hai cách trực tiếp và gián tiếp.
- Câu 5: Cho con trỏ p1 trỏ đến phần tử thứ 3 còn con trỏ p2 trỏ đến phần tử thứ 4 của mảng int. Hỏi $p2 - p1 = ?$



Bài tập

Bài 6: Cho đoạn lệnh sau

```
#include<iostream>  
  
using namespace std;  
  
int main() {  
    float pay, *ptr_pay;  
    pay = 2313.54;  
    ptr_pay = &pay;  
    return 0;  
}
```

Viết chương trình trên vào máy và chạy debug và xem các giá trị thực tế của: pay, *ptr_pay, &ptr_pay, *pay, &pay trước khi chạy lệnh return 0.

Bài 7: Tìm lỗi trong đoạn code sau:

```
#include<iostream>  
  
using namespace std;  
  
int main() {  
    int *x, y = 2;  
    *x = y;  
    *x += y++;  
    cout << *x << y;  
}
```



Chúc các em học tốt !

