



# IT001 - NHẬP MÔN LẬP TRÌNH

## CHƯƠNG 8: CON TRỎ (TT)

Cấp phát động (dynamic memory allocation) là một tính năng thiết yếu trong lập trình C++ giúp ta quản lý bộ nhớ linh hoạt và hiệu quả. Khác với cấp phát tĩnh (static memory allocation), cấp phát động cho phép ta phân bổ vùng nhớ khi cần thiết trong quá trình thực thi chương trình, tối ưu hóa việc sử dụng bộ nhớ và xây dựng các cấu trúc dữ liệu động mạnh mẽ.

Khoa Khoa học Máy tính



# IT001 - NHẬP MÔN LẬP TRÌNH

## CHƯƠNG 8.2: CON TRỎ CẤP PHÁT ĐỘNG

Cấp phát động (dynamic memory allocation) là một tính năng thiết yếu trong lập trình C++ giúp ta quản lý bộ nhớ linh hoạt và hiệu quả. Khác với cấp phát tĩnh (static memory allocation), cấp phát động cho phép ta phân bổ vùng nhớ khi cần thiết trong quá trình thực thi chương trình, tối ưu hóa việc sử dụng bộ nhớ và xây dựng các cấu trúc dữ liệu động mạnh mẽ.

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

- 8.11 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



## **8.11 Cấp phát và giải phóng ô nhớ**



## 8.11 Cấp phát và giải phóng ô nhớ

8.11.1 Quản lý bộ nhớ trong C++

8.11.2 Cấp phát tĩnh, cấp phát động

8.11.3 Biến cấp phát tĩnh và Biến cấp phát động

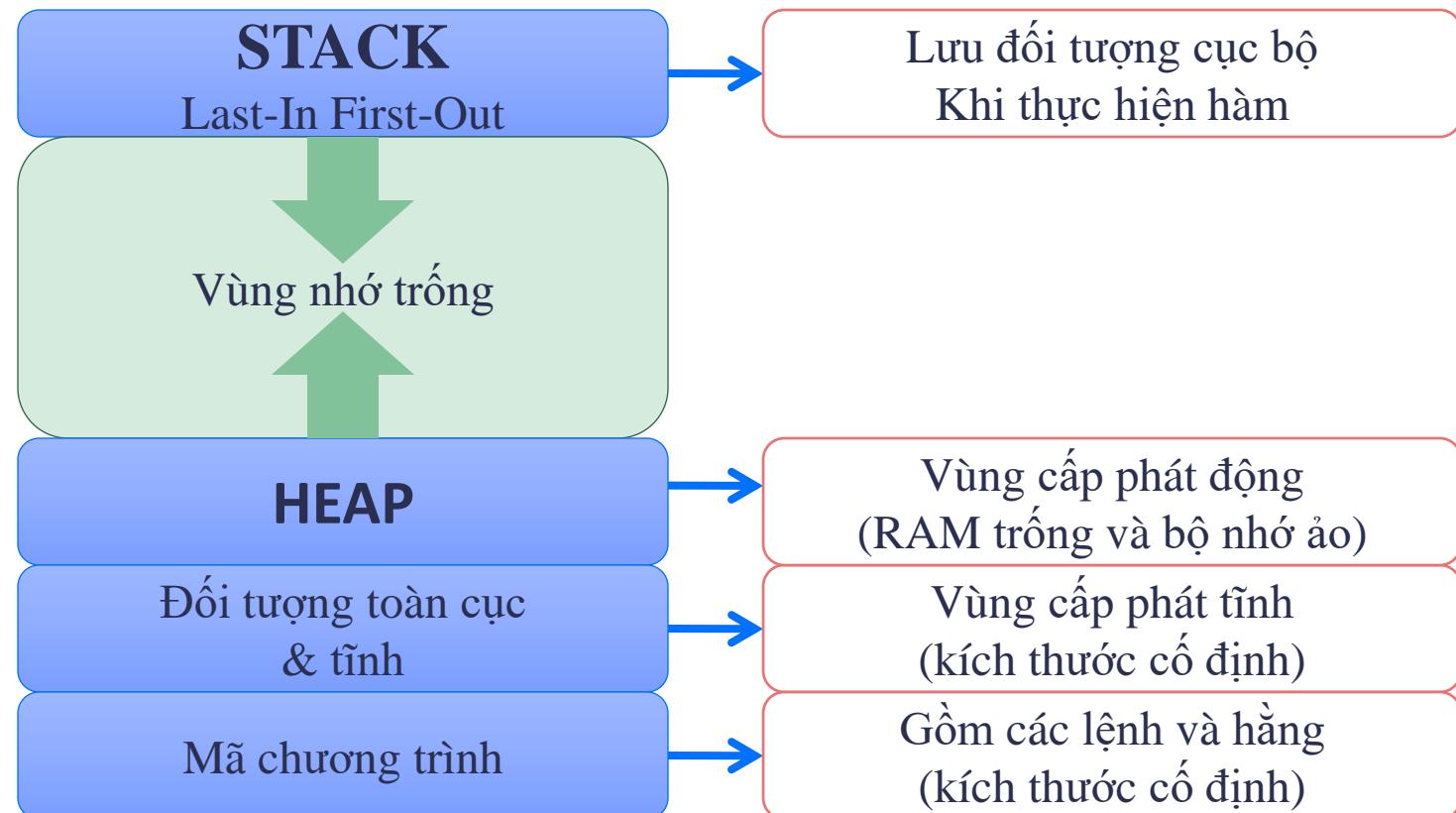
8.11.4 Toán tử cấp phát động **new**

8.11.5 Toán tử giải phóng ô nhớ **delete**



## 8.11.1 Quản lý bộ nhớ trong C++

- Toàn bộ tập tin chương trình sẽ được nạp vào bộ nhớ tại vùng nhớ còn trống, gồm 4 phần:





## 8.11.2 Cấp phát tĩnh, cấp phát động

- Cấp phát tĩnh (Static memory allocation và Automatic memory allocation)
  - Khai báo biến, cấu trúc, mảng, ...
  - Bắt buộc phải biết trước cần bao nhiêu bộ nhớ lưu trữ
  - Không thay đổi được kích thước suốt chương trình
  - Tồn tại trong suốt thời gian tồn tại của chương trình
- Cấp phát động (dynamic memory allocation)
  - Cần bao nhiêu cấp phát bấy nhiêu
  - Có thể giải phóng nếu không cần sử dụng
  - Sử dụng vùng nhớ dành cho cấp phát động



## 8.11.2 Cấp phát tĩnh, cấp phát động

- **Cấp phát bộ nhớ**

- Trong C: Hàm **malloc**, **calloc**, **realloc** (**<stdlib.h>** hoặc **<alloc.h>**)
- Trong C++: Toán tử **new**

- **Giải phóng bộ nhớ**

- Trong C: Hàm **free**
- Trong C++: Toán tử **delete**



## 8.11.3 Biến cấp phát tĩnh và Biến cấp phát động

- **Biến cục bộ (automatic variable)**

- Khai báo bên trong định nghĩa hàm
- Sinh ra khi hàm được gọi
- Hủy đi khi hàm kết thúc
- Biến cục bộ được đặt tên
- Thường gọi là biến tự động (automatic variable) nghĩa là được trình biên dịch quản lý một cách tự động

- **Biến cấp phát động (dynamic variable)**

- Sinh ra bởi cấp phát động
- Sinh ra và hủy đi khi chương trình đang chạy
- Vùng nhớ cấp phát động không có tên gọi
- Biến cấp phát động hay Biến động là biến con trả trước khi sử dụng **được cấp phát bộ nhớ**.



## 8.11.4 Toán tử cấp phát động new

- Có thể cấp phát động cho biến con trỏ bằng toán tử **new**.
- Toán tử **new** sẽ tạo ra vùng nhớ “không tên” cho con trỏ trả về.

- **Cú pháp:**

```
<type> *pointerName = new <type>
```

```
<type> pointer = new <type> (value)
```

- **Ví dụ:**

```
int *ptr = new int;
```

```
int *ptr1 = new int(100); // (*ptr)=100
```

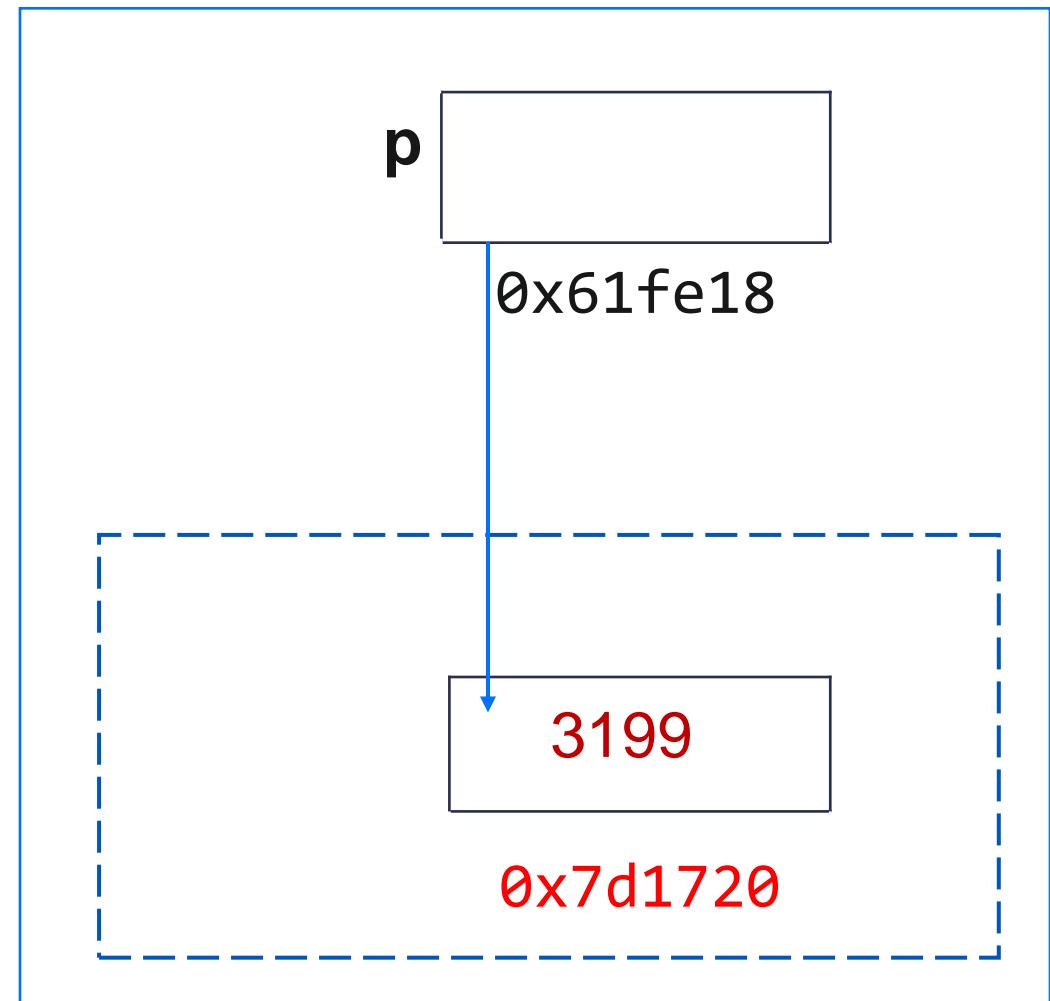
•



## 8.11.4 Toán tử cấp phát động new

- Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    ▶ int *p;
    ▶ p = new int;
    ▶ if (p == nullptr) {
        cout << "Error: Khong du bo nho.\n";
        exit(1);
    }
    ▶ *p = 3199;
    ▶ return 0;
}
```



Memory Layout



# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
int main() {
    int *p;
    cout << "p: " << p << endl;
    cout << "&p: " << &p << endl;
    p = new int;
    cout << "\nSau cap phat => p: " << p << endl;
    if (p == nullptr) {
        cout << "Error: Khong du bo nho.\n";
        exit(1);
    }
    *p = 3199;
    cout << "*p: " << *p << endl;
    return 0;
}
```

```
p: 0x10
&p: 0x61fe18
Sau cap phat => p: 0x721720
*p: 3199
```



## 8.11.4 Toán tử cấp phát động new

- Chạy đoạn lệnh sau:

1. `int *p1, *p2;`
2. `int *p1 = new int;`
3. `*p1 = 30;`
4. `p2 = p1;`
5. `*p2 = 40;`
6. `p1 = new int;`
7. `*p1 = 50;`

- Dòng 1. `int *p1, *p2;`



- Dòng 2. `int *p1 = new int;`



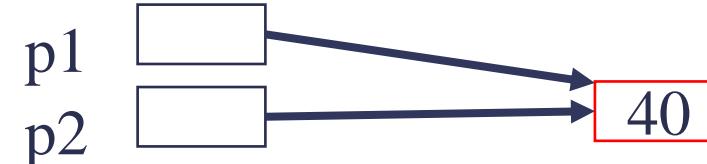
- Dòng 3. `*p1 = 30;`



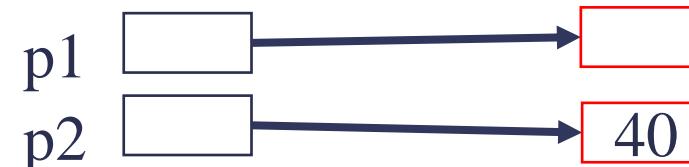
- Dòng 4. `p2 = p1;`



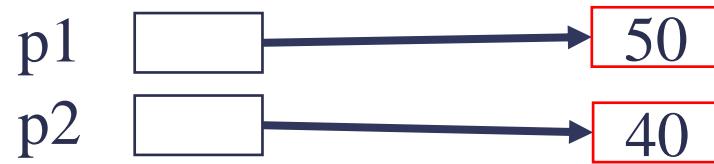
- Dòng 5. `*p2 = 40;`



- Dòng 6. `p1 = new int;`



- Dòng 7. `*p1 = 50;`





## 8.11.5 Toán tử giải phóng ô nhớ delete

- Toán tử **delete** dùng để giải phóng vùng nhớ trong HEAP do con trỏ trả về (con trỏ được cấp phép bằng toán tử new).
- Cú pháp: **delete pointerName;**
- Lưu ý: Sau khi gọi toán tử **delete** thì con trỏ vẫn trả về vùng nhớ trước khi gọi hàm delete. Ta gọi là “con trỏ lạc”. Ta vẫn có thể gọi tham chiếu trên con trỏ, tuy nhiên:
  - Kết quả không lường trước được
  - Thường là nguy hiểm⇒ Hãy tránh con trỏ lạc bằng cách gán con trỏ bằng **nullptr** sau khi delete.

Ví dụ: **delete pointer;**

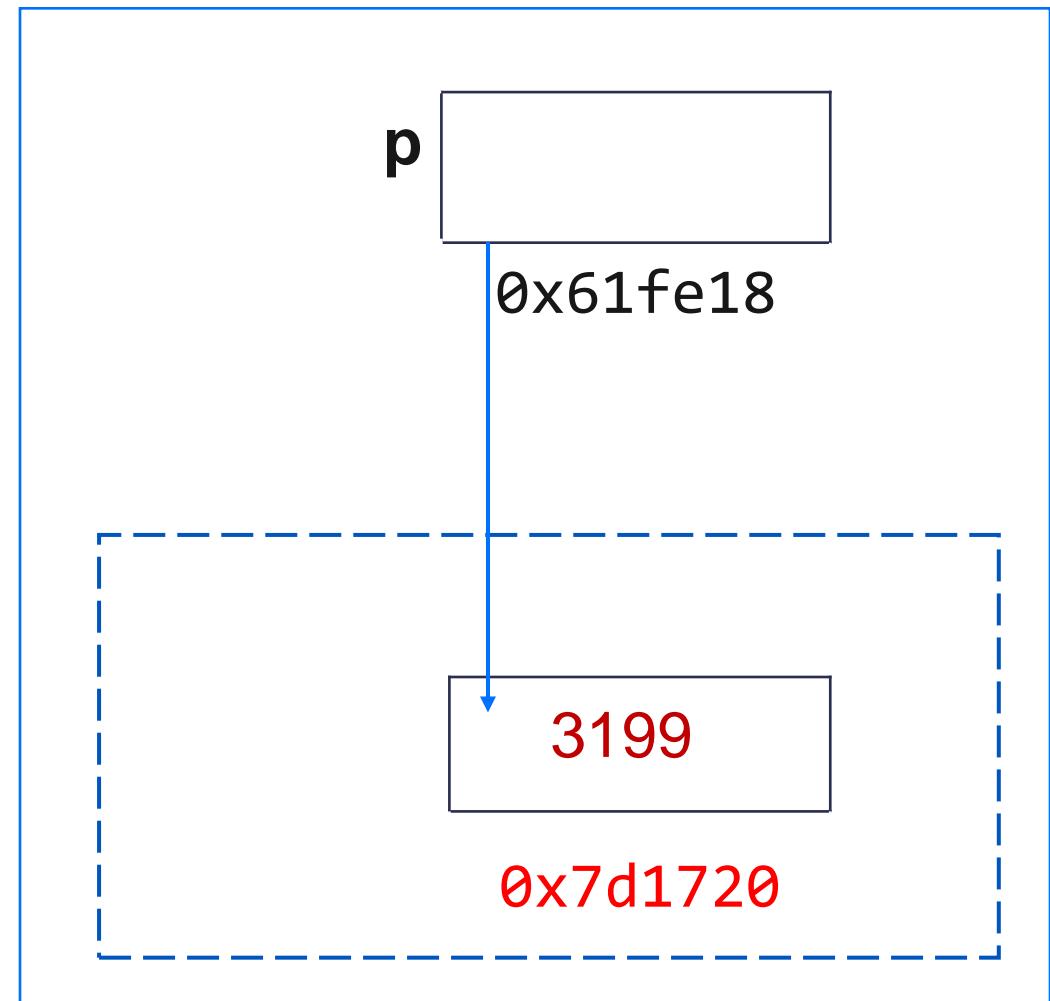
**pointer = nullptr;**



## 8.11.5 Toán tử giải phóng ô nhớ delete

- Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    int *p;
    p = new int;
    *p = 3199;
    delete p;
    p = nullptr;
    return 0;
}
```



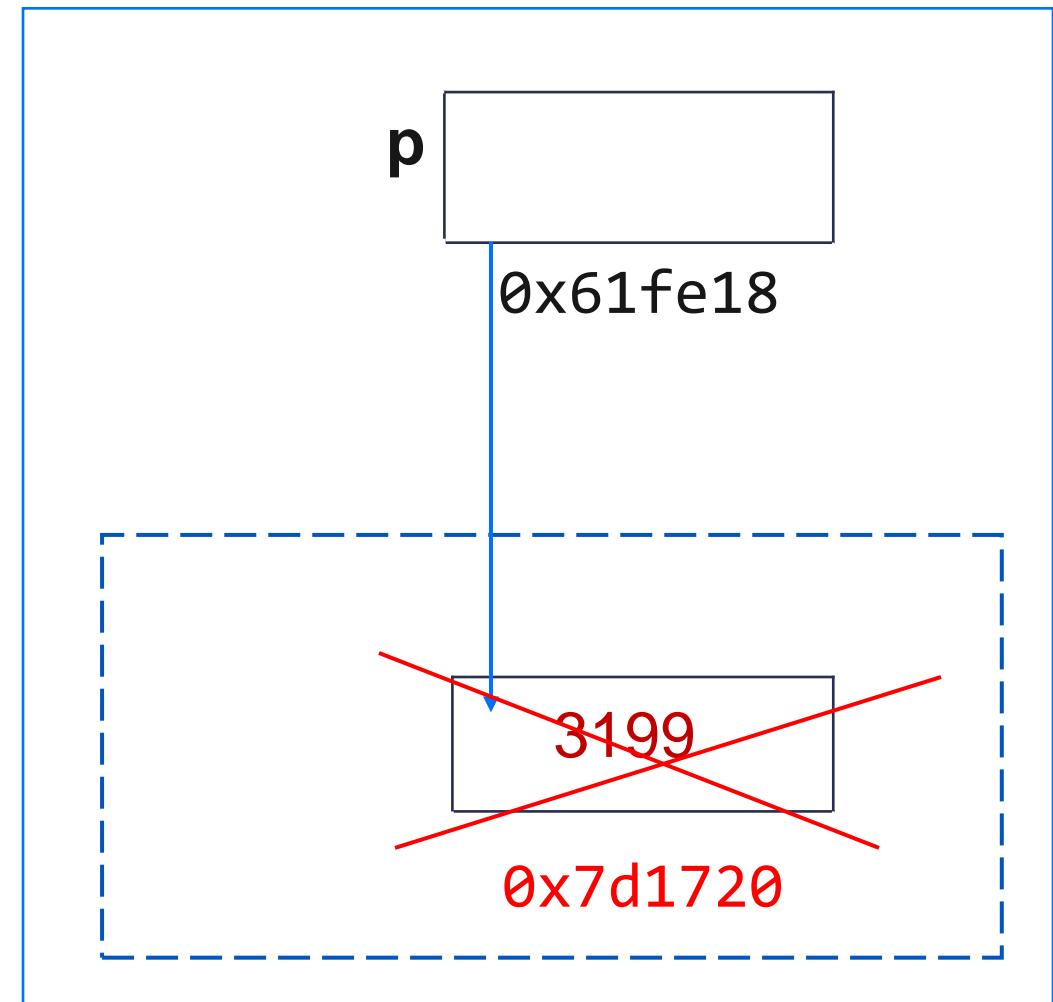
Memory Layout



## 8.11.5 Toán tử giải phóng ô nhớ delete

- Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    int *p;
    p = new int;
    *p = 3199;
    delete p;
    p = nullptr;
    return 0;
}
```



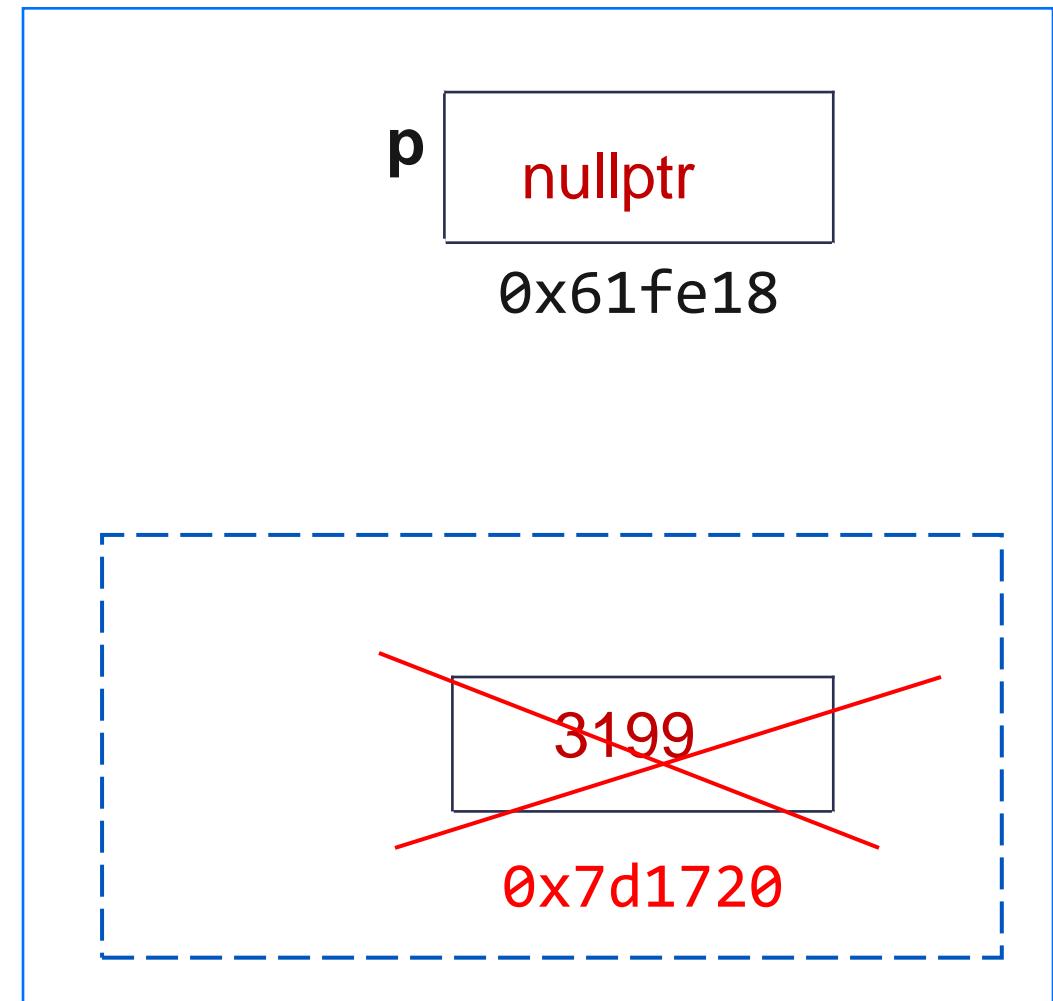
Memory Layout



## 8.11.5 Toán tử giải phóng ô nhớ delete

- Ví dụ:

```
#include <iostream>
using namespace std;
int main() {
    int *p;
    p = new int;
    *p = 3199;
    delete p;
    p = nullptr;
    return 0;
}
```



Memory Layout



## 8.11.5 Toán tử giải phóng ô nhớ delete

- Không cần kiểm tra con trỏ có `nullptr` hay không trước khi giải phóng vùng nhớ cấp phát động.
- Cấp phát bằng `malloc`, `calloc` hay `realloc` thì giải phóng bằng `free`
- Cấp phát bằng `new` thì giải phóng bằng `delete`, cấp phát mảng bằng `new []` thì giải phóng bằng `delete []`.



## 8.12 Mảng một chiều cấp phát động



# Hạn chế của mảng tĩnh

- Kích thước phân bổ bộ nhớ tại thời điểm biên dịch → Cần phải biết trước kích thước của mảng
- Không thay đổi được kích thước mảng → Trong quá trình chạy nếu dùng ít hơn (kích thước mảng đang có) thì sẽ lãng phí, nếu dùng nhiều hơn thì không đủ

## → Mảng động

- Không cần xác định trước kích thước tại thời điểm lập trình
- **Có thể cấp phát và giải phóng bộ nhớ trong quá trình thực thi**



# Tạo mảng động bằng toán tử new

- Cấp phát động cho biến con trỏ
- Sau đó dùng con trỏ như mảng chuẩn
- Cú pháp:
- Ví dụ:

```
type pointer = new type [number_of_elements]
```

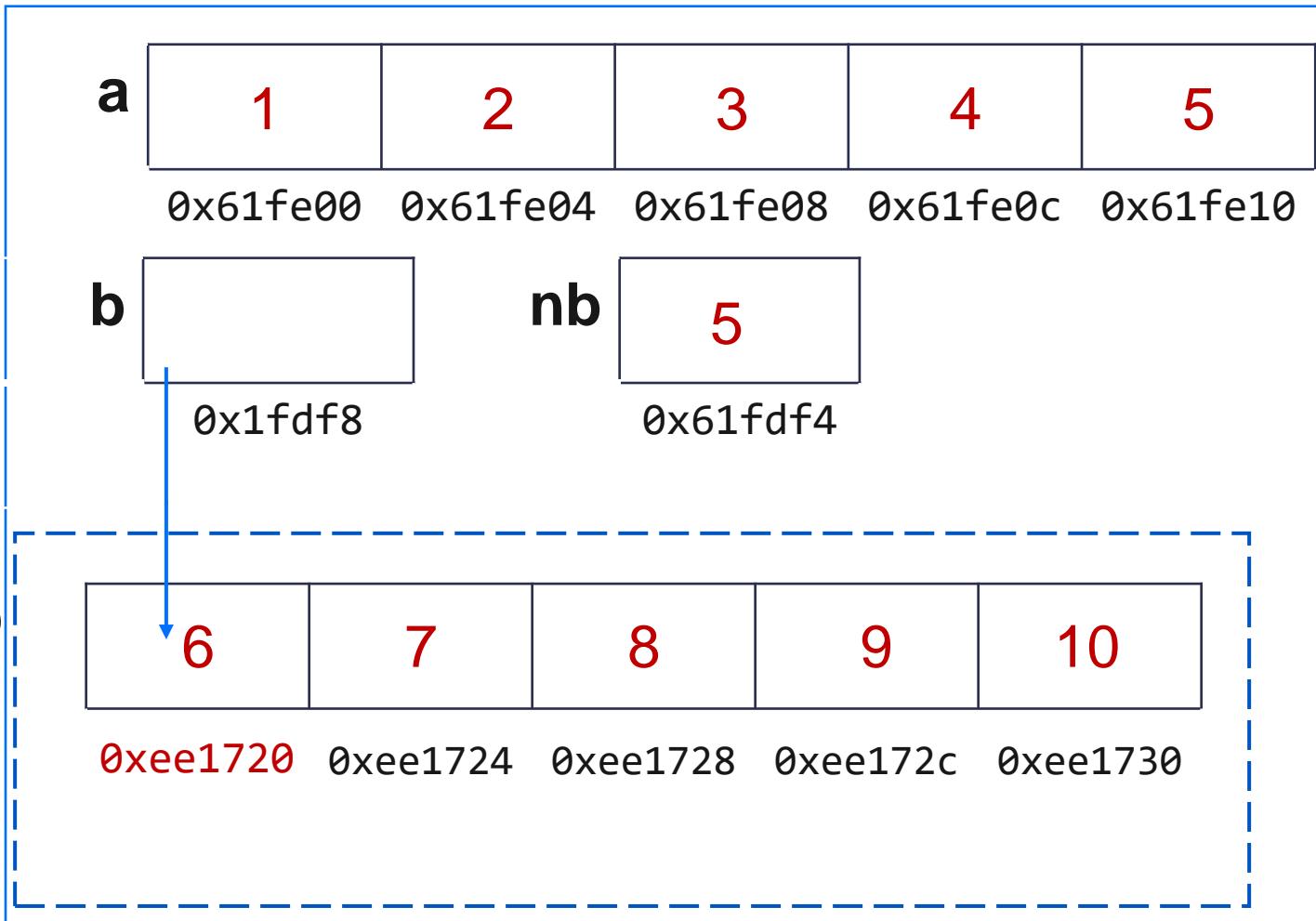
```
// typedef double * doublePtr;  
// doublePtr d;  
double* d;  
d = new double[10];
```

⇒ Tạo biến mảng cấp phát động d có 10 phần tử, kiểu cơ sở là double.



# Minh họa cấp phát động mảng 1 chiều

```
#include <iostream>
using namespace std;
int main() {
    ▶ int a[5] = {1, 2, 3, 4, 5};
    ▶ int *b, nb;
    ▶ nb = 5;
    ▶ b = new int[nb];
    ▶ for (int i = 0; i < nb; i++)
        cin >> b[i];
    ▶ return 0;
}
```





# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
int main() {
    int a[5];
    cout << "a : " << a << endl;
    for (int i = 0; i < 5; i++)
        cout << "&a[" << i << "] : " << &a[i] << endl;
    cout << endl;
    int *b, nb;
    cout << "&nb: " << &nb << endl;
    cout << "b : " << b << endl;
    cout << "&b : " << &b << endl;
    nb = 5;
    b = new int[nb];
    cout << "\nSau cap phat => b: " << b << endl;
    for (int i = 0; i < nb; i++)
        cout << "&b[" << i << "] : " << &b[i] << endl;
    return 0;
}
```

a : 0x61fe00  
&a[0]: 0x61fe00  
&a[1]: 0x61fe04  
&a[2]: 0x61fe08  
&a[3]: 0x61fe0c  
&a[4]: 0x61fe10

&nb: 0x61fdf4  
b : 0x401519  
&b : 0x61fdf8

Sau cap phat => b: 0xe81720  
&b[0]: 0xe81720  
&b[1]: 0xe81724  
&b[2]: 0xe81728  
&b[3]: 0xe8172c  
&b[4]: 0xe81730



# Xóa mảng động

- Dùng toán tử `delete[]` để xóa mảng động.

Ví dụ:

```
double *d = new double[10];  
//... Processing  
delete[] d;
```

- ⇒ Giải phóng tất cả vùng nhớ của mảng động này
- ⇒ Cặp ngoặc vuông báo hiệu có mảng
- ⇒ Nhắc lại: `d` vẫn trỏ tới vùng nhớ đó. Vì vậy sau khi `delete`, cần gán: `d = nullptr`;



## 8.13 Con trỏ cấp phát động và chuỗi



## 8.13 Con trỏ cấp phát động và chuỗi

- Chuỗi C-String cũng là mảng 1 chiều, nên có thể dùng cấp phát động để quản lý chuỗi.
- Ví dụ: **Hỏi:** Cách khai báo và sử dụng biến chuỗi dưới đây có đúng không?

```
char s1[100];  
s1 = "hello.";      → SAI
```

```
char *s2;  
s2 = "hello.";      → ĐÚNG
```

```
char *s3 = new char[100];  
strcpy(s3, "hello.");      → ĐÚNG
```



## 8.13 Con trỏ cấp phát động và chuỗi

- Cho các lệnh sau:

1. `char s1[] = "hello.;"`
2. `char *s2 = "hello.;"`
3. `char *s3 = new char[100];`  
`strcpy(s3, "hello.");`
4. `s1[0] = 'H';` → ĐÚNG
5. `s2[0] = 'H';` → SAI
6. `s3[0] = 'H';` → ĐÚNG

- Hỏi lệnh nào trên là chưa đúng?



## 8.14 Mảng hai chiều cấp phát động



## 8.14 Mảng hai chiều cấp phát động

- Ví dụ khai báo mảng 3 dòng 4 cột:

- Khai báo mảng các con trỏ:

```
typedef int* IntArrayPtr;  
IntArrayPtr *m = new IntArrayPtr[3];
```

Tương đương: `int **m = new int*[3];`

⇒ Tạo ra mảng 3 con trỏ

⇒ Sau đó cấp phát cho mỗi con trỏ này quản lý một mảng 4 biến kiểu int:

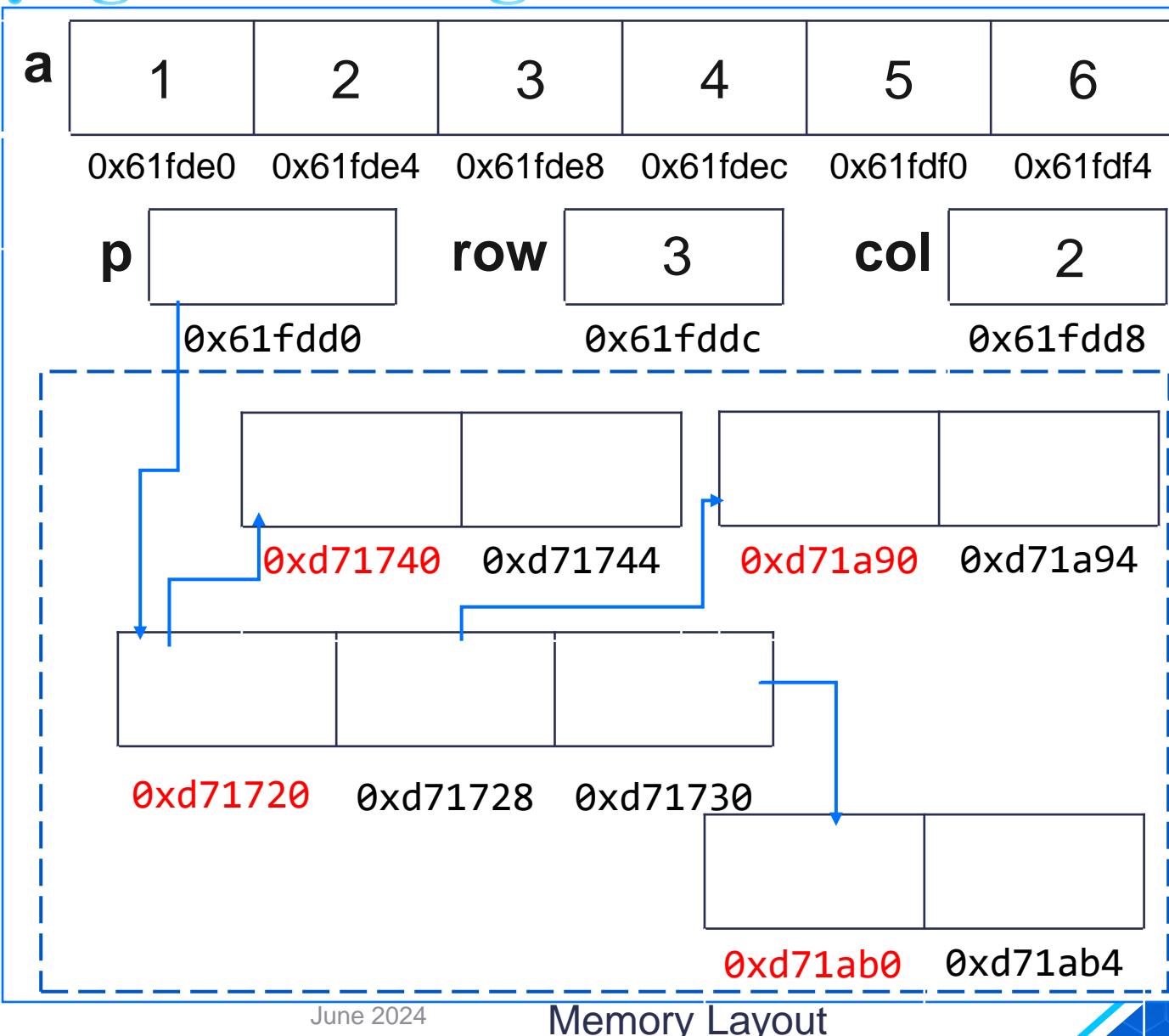
```
for (int i = 0; i < 3; i++)  
    m[i] = new int[4];
```

⇒ Kết quả là mảng động  $3 \times 4$



# Minh họa cấp phát động cho mảng 2 chiều

- ▶ `int a[3][2]={1, 2, 3, 4, 5, 6};`
- ▶ `int **p, row=3, col=2;`
- ▶ `p = new int*[row];`
- ▶ `for (int i = 0; i < row; i++)`
- ▶  `p[i] = new int[col];`





# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
int main() {
    int a[3][2] = {1, 2, 3, 4, 5, 6};
    cout << "Dia chi cac phan tu cua mang a[3][2]: " << endl;
    for(int i=0; i<3; i++){
        cout << "Dong " << i << ": ";
        for(int j=0; j<2; j++)
            cout << &a[i][j] << " ";
        cout << endl;
    }
    int row=3, col=2;
    cout << "\nrow: " << &row << " col: " << &col << endl;
    int **p;
    cout << "p : " << p << endl;
    cout << "&p: " << &p << endl;
    p = new int*[row];
    cout << "\nSau khi cap phap => p: " << p << endl;
    cout << "&p[0]: " << &p[0] << endl;
    cout << "&p[1]: " << &p[1] << endl;
    cout << "&p[2]: " << &p[2] << endl;
    cout << "Dia chi cac phan tu cua mang p: " << endl;
    for (int i = 0; i < row; i++) {
        p[i] = new int[col];
        cout << "Dong " << i << ":" 
            << &p[i][0] << " " << &p[i][1] << endl;
    }
    return 0;
}
```

Dia chi cac phan tu cua mang a[3][2]:  
Dong 0: 0x61fde0 0x61fde4  
Dong 1: 0x61fde8 0x61fdec  
Dong 2: 0x61fdf0 0x61fdf4  
  
row: 0x61fddc col: 0x61fdd8  
p : 0x4019f0  
&p: 0x61fdd0  
  
Sau khi cap phap => p: 0x1051720  
&p[0]: 0x1051720  
&p[1]: 0x1051728  
&p[2]: 0x1051730  
Dia chi cac phan tu cua mang p:  
Dong 0: 0x1051740 0x1051744  
Dong 1: 0x1051a90 0x1051a94  
Dong 2: 0x1051ab0 0x1051ab4



# Xóa mảng động 2 chiều

- Dùng toán tử `delete[]` để xóa mảng động. Ví dụ:

```
double **pd;  
// ... Processing  
for (int i = 0; i < row; i++)  
    delete[] p[i];  
delete[] p;
```

- ⇒ Giải phóng tất cả vùng nhớ của mảng động này
- ⇒ Cặp ngoặc vuông báo hiệu có mảng
- ⇒ Nhắc lại: d vẫn trỏ tới vùng nhớ đó.
- ⇒ Vì vậy sau khi `delete`, cần gán: `d = nullptr`;



## 8.15 Con trỏ và hàm số



## 8.15 Con trỏ cấp phát động và hàm số

8.15.1 Hàm truyền tham số là con trỏ

8.15.2 Hàm truyền tham số là tham chiếu con trỏ

8.15.3 Hàm trả về con trỏ cấp phát động



## 8.15 Con trỏ và hàm số

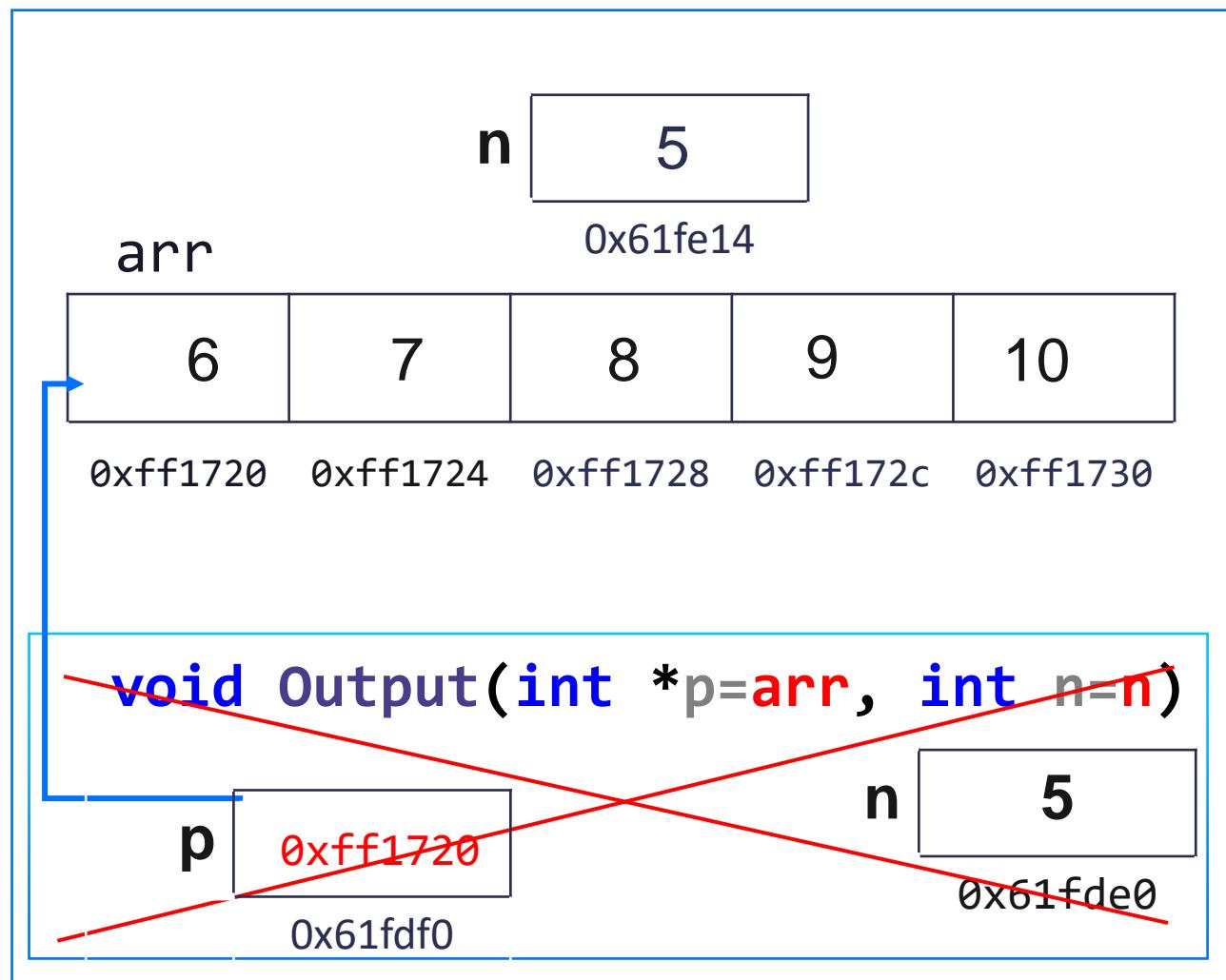
### 8.15.1 Hàm truyền tham số là con trỏ



## 8.15.1 Hàm truyền tham số là con trỏ

```
// Hàm xuất mảng
void Output(int *p, int n) {
    cout << "\n Xuat mang 1 chieu: ";
    for (int i = 0; i < n; i++)
        cout << p[i] << " ";
}

int main() {
    int arr[] = {6, 7, 8, 9, 10}, n=5;
    Output(arr, n);
    return 0;
}
```





# Code minh họa hiển thị giá trị các biến

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

void Output(int *p, int n) {
    cout << "\nTrong ham Output: " << endl;
    cout << "n: " << n << " -- &n: " << &n << endl;
    cout << "p: " << p << " -- &p: " << &p << endl;
    cout << "\nXuat mang 1 chieu: ";
    for (int i = 0; i < n; i++)
        cout << p[i] << " ";
}

int* Input(int n) {
    int *p = new int[n];
    cout << "Nhập " << n << " phần tử mảng: ";
    for (int i = 0; i < n; i++)
        cin >> p[i];
    return p;
}

int main() {
    int *arr, n;
    cout << "Nhập n: ";
    cin >> n;
    arr = Input(n);
    cout << "\nTrước khi vào Output: " << endl;
    cout << "n : " << n << " -- &n : " << &n << endl;
    cout << "arr: " << arr << " -- &arr: " << &arr << endl;
    Output(arr, n);
    return 0;
}
```

```
Nhập n: 5
Nhập 5 phần tử mảng: 1 2 3 4 5

Trước khi vào Output:
n : 5 — &n : 0x61fe14
arr: 0xe11720 — &arr: 0x61fe18

Trong ham Output:
n: 5 — &n: 0x61fdf8
p: 0xe11720 — &p: 0x61fdf0

Xuất mảng 1 chiều: 1 2 3 4 5
```



## 8.15.1 Hàm truyền tham số là con trỏ

- Xét 2 đoạn chương trình sau. Tìm lỗi sai và giải thích.

```
int main() {  
    int a[] = {1, 2, 3, 4, 5, 6};  
  
    for (int i = 0; i<6; i++)  
        printf("%d", *(a++));  
}
```

SAI

```
void xuat(int *a, int n) {  
    for (int i = 0; i<n; i++)  
        printf("%d", *(a++));  
}  
  
int main() {  
    int a[] = {1, 2, 3, 4, 5, 6};  
    xuat(a, 6);  
}
```

ĐÚNG

Lý do: Đối số mảng truyền cho hàm không phải hằng con trỏ.



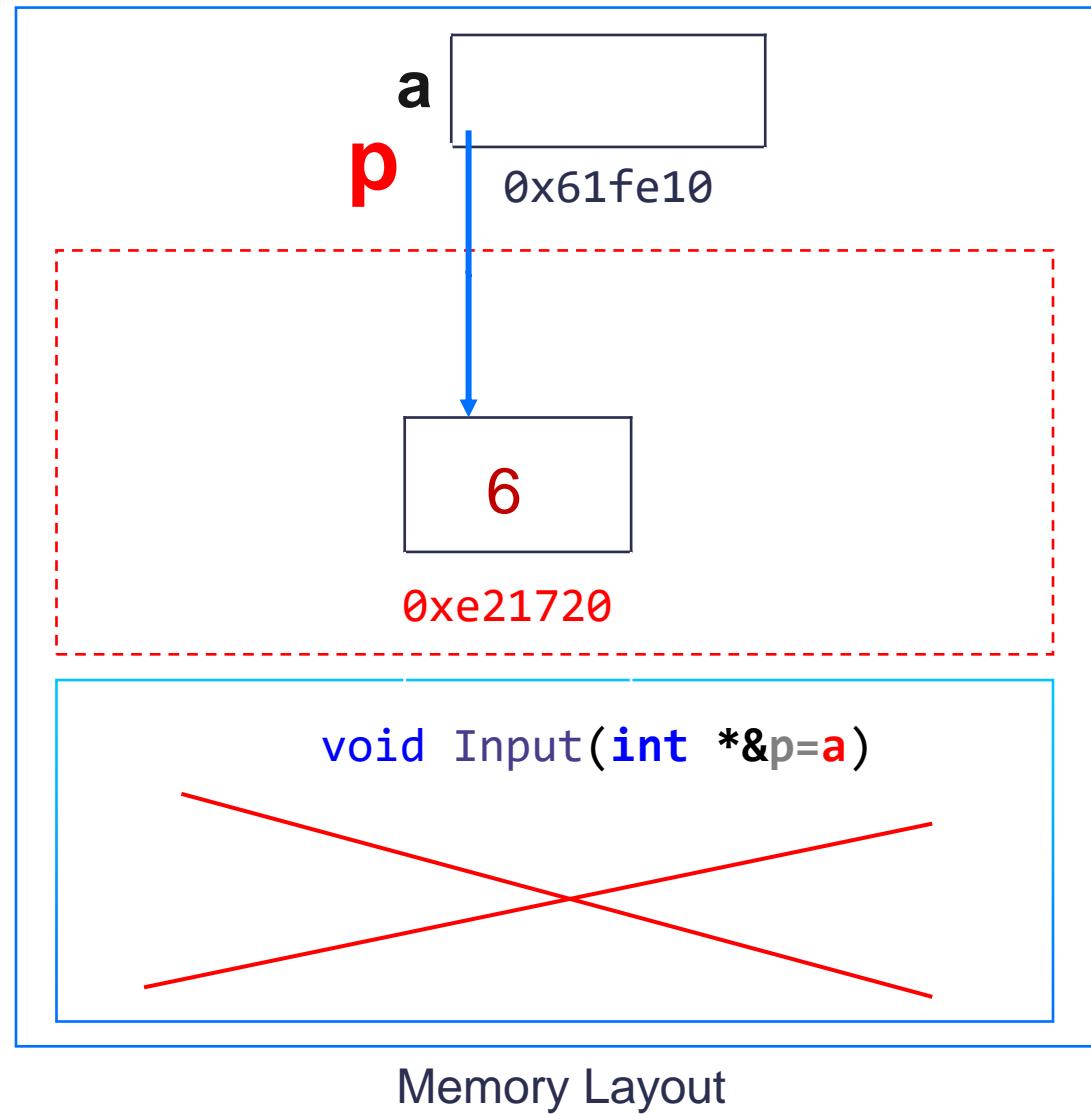
## 8.15 Con trỏ và hàm số

### 8.15.2 Hàm truyền tham số là tham chiếu con trỏ



# Ví dụ 1: Hàm khai báo tham chiếu con trỏ quản lý 1 vùng nhớ

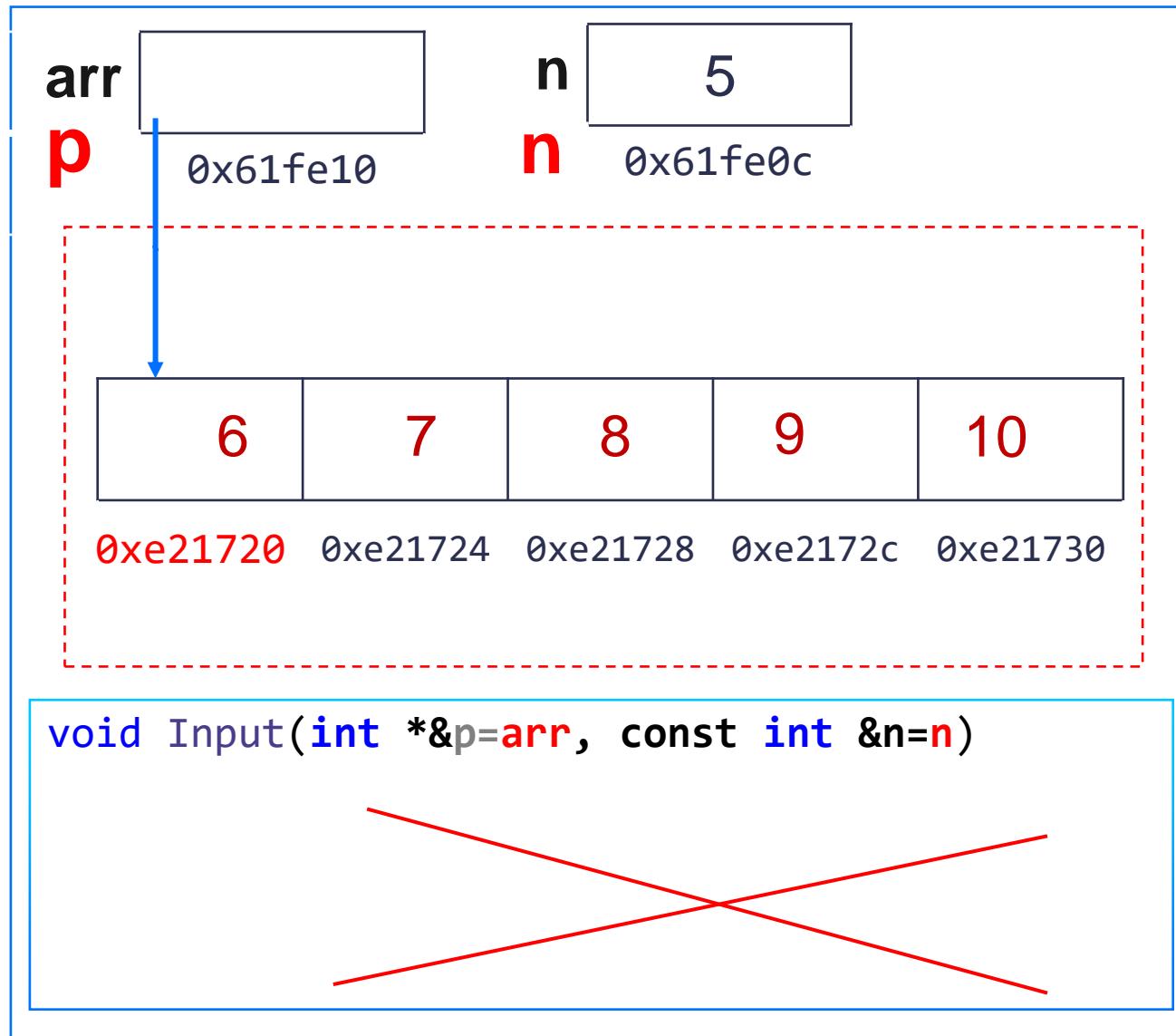
```
#include <iostream>
using namespace std;
void Input(int *&p) {
    p = new int;
    cin >> *p;
}
int main() {
    int *a;
    Input(a);
    return 0;
}
```





## Ví dụ 2: Hàm khai báo tham chiếu con trỏ quản lý mảng 1D

```
#include <iostream>
using namespace std;
void Input(int *&p, const int &n){
    p = new int[n];
    for (int i = 0; i < n; i++)
        cin >> p[i];
}
int main() {
    int *arr, n;
    cout << "Nhập n: ";
    cin >> n;
    Input(arr, n);
    return 0;
}
```





# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
void Input(int *&p, int n) {
    cout << "\nTrong hàm Input: " << endl;
    cout << "&n: " << &n << endl;
    cout << "&p: " << &p << endl;
    p = new int[n];
    cout << "Sau cap phat => p: " << p << endl;
    cout << "Nhập " << n << " phần tử mảng: ";
    for (int i = 0; i < n; i++)
        cin >> p[i];
    for (int i = 0; i < n; i++)
        cout << "&p[" << i << "]:" << p+i << endl;
}
int main() {
    int *arr, n;
    cout << "Nhập n: ";
    cin >> n;
    cout << "&n: " << &n << endl;
    cout << "&arr: " << &arr << endl;
    Input(arr, n);

    cout << "\nRa ngoài hàm Input: " << endl;
    for (int i = 0; i < n; i++)
        cout << "&arr[" << i << "]:" << arr+i << endl;
    return 0;
}
```

```
Nhập n: 5
&n: 0x61fe0c
&arr: 0x61fe10

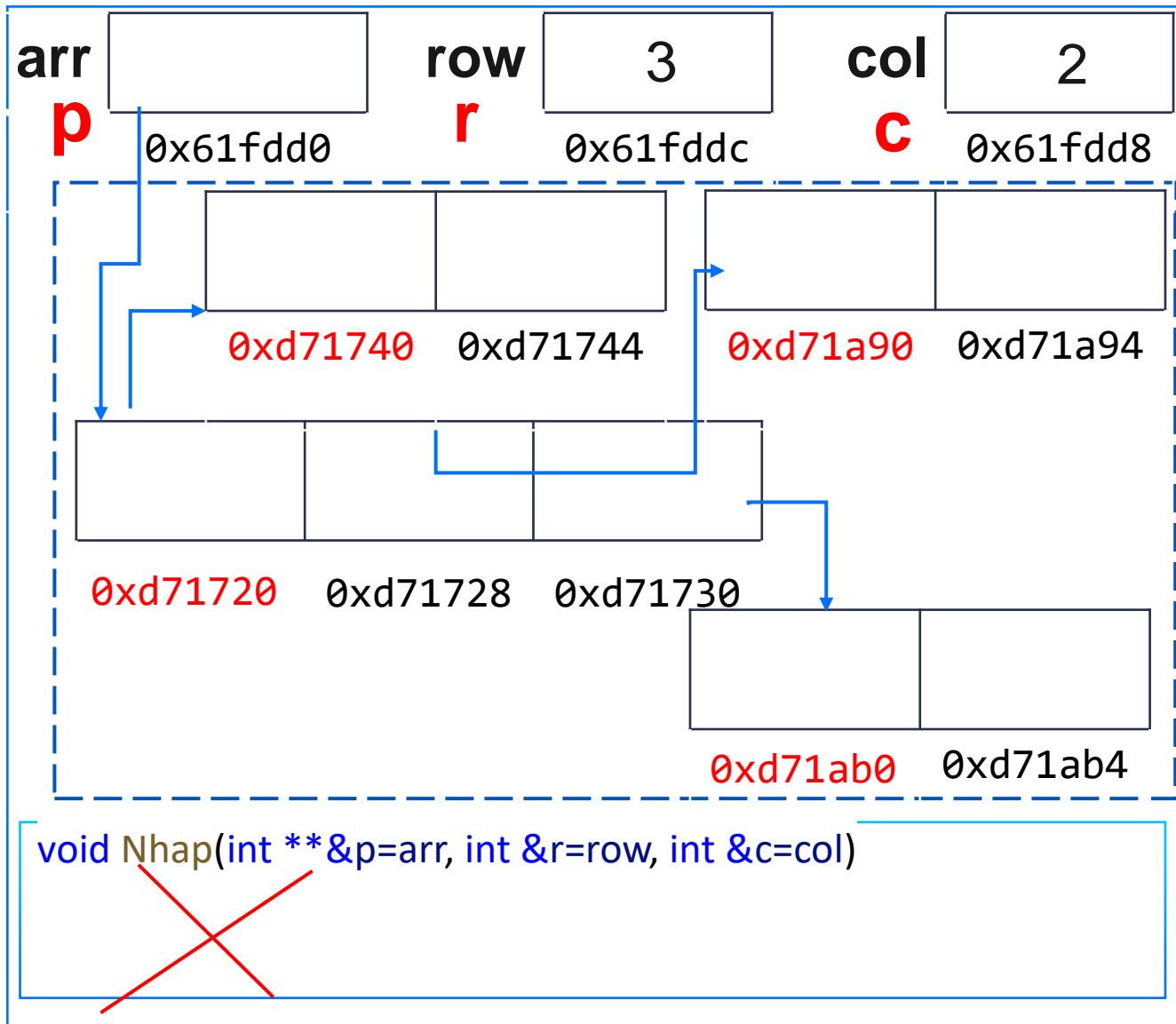
Trong hàm Input:
&n: 0x61fde8
&p: 0x61fe10
Sau cap phat => p: 0x7a1720
Nhập 5 phần tử mảng: 1 2 3 4 5
&p[0]: 0x7a1720
&p[1]: 0x7a1724
&p[2]: 0x7a1728
&p[3]: 0x7a172c
&p[4]: 0x7a1730

Ra ngoài hàm Input:
&arr[0]: 0x7a1720
&arr[1]: 0x7a1724
&arr[2]: 0x7a1728
&arr[3]: 0x7a172c
&arr[4]: 0x7a1730
```



# Ví dụ: Hàm khai báo tham chiếu con trỏ quản lý mảng 2D

```
#include <iostream>
using namespace std;
void Nhap(int **&p, int &r, int &c){
    cin >> r >> c;
    p = new int*[r];
    for (int i = 0; i < r; i++)
        p[i] = new int[c];
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            cin >> p[i][j];
}
int main() {
    int **arr, row, col;
    Nhap(arr, row, col);
    return 0;
}
```





# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
void Nhap(int **&p, int &r, int &c){
    cout << "\nVao ham Nhap: " << endl;
    cout << "Nhap dong, cot: ";
    cin >> r >> c;
    p = new int*[r];
    cout << "\nSau khi cap phap => p: " << p << endl;
    cout << "\t&p[0]: " << &p[0] << endl;
    cout << "\t&p[1]: " << &p[1] << endl;
    cout << "\t&p[2]: " << &p[2] << endl;
    cout << "\n\tDia chi cac phan tu mang: " << endl;
    for (int i = 0; i < r; i++) {
        p[i] = new int[c];
        cout << "\t" << &p[i][0] << " " << &p[i][1] << endl;
    }
}
void Xuat(int **p, const int &r, const int &c){
    cout << "\nVao ham Xuat: " << endl;
    cout << "Dia chi cac phan tu mang: " << endl;
    for (int i = 0; i < r; i++){
        cout << "\t";
        for (int j = 0; j < c; j++)
            cout << &p[i][j] << " ";
        cout << endl;
    }
}
int main() {
    int **arr, row, col;
    cout << "arr : " << arr << endl;
    cout << "&arr: " << &arr << endl;
    Nhap(arr, row, col);
    Xuat(arr, row, col);
    return 0;
}
```

arr : 0x10  
&arr: 0x61fe18

Vao ham Nhap:  
Nhap dong, cot: 3 2

Sau khi cap phap => p: 0x1001720  
&p[0]: 0x1001720  
&p[1]: 0x1001728  
&p[2]: 0x1001730

Dia chi cac phan tu mang:  
0x1001740 0x1001744  
0x1001a90 0x1001a94  
0x1001ab0 0x1001ab4

Vao ham Xuat:  
Dia chi cac phan tu mang:  
0x1001740 0x1001744  
0x1001a90 0x1001a94  
0x1001ab0 0x1001ab4



## 8.15 Con trỏ và hàm số

### 8.15.3 Hàm trả về con trỏ cấp phát động



## 8.15.3 Hàm trả về con trỏ cấp phát động

- Ta không được phép trả về kiểu mảng trong hàm.

Ví dụ:

```
int[] someFunction(); // KHÔNG HỢP LỆ
```

- Có thể thay bằng trả về con trỏ tới mảng có cùng kiểu cơ sở:

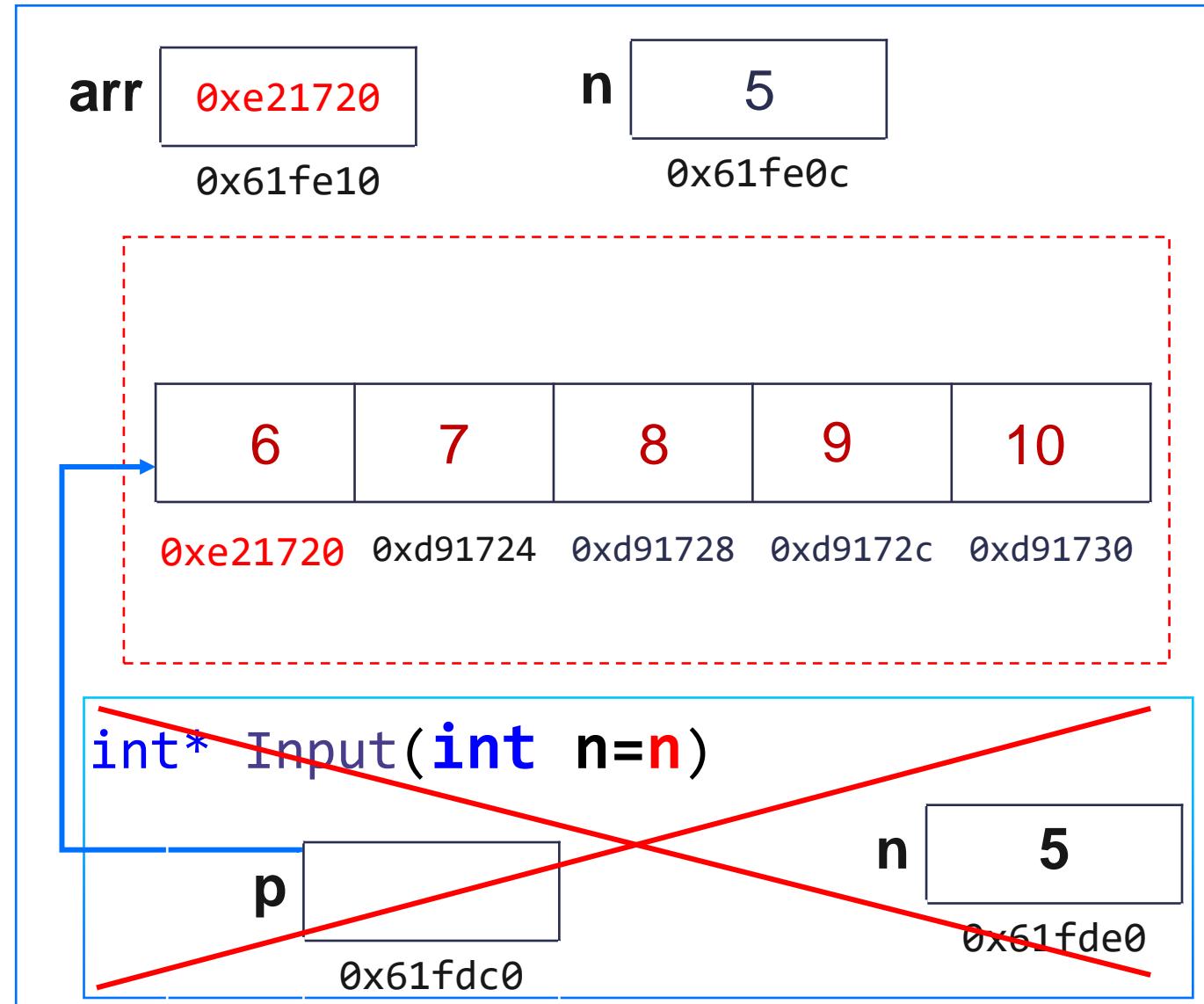
```
int* someFunction(); // HỢP LỆ
```



## 8.15.3 Hàm trả về con trỏ cấp phát động

- Ví dụ:

```
#include <iostream>
using namespace std;
int* Input(int n) {
    int *p = new int[n];
    for (int i = 0; i < n; i++)
        cin >> p[i];
    return p;
}
int main() {
    int *arr, n;
    cout << "Nhập n: "; cin >> n;
    arr = Input(n);
    return 0;
}
```





# Code minh họa hiển thị giá trị các biến

```
#include <iostream>
using namespace std;
int* Input(int n) {
    int *p = new int[n];
    cout << "\nTrong ham Input: " << endl;
    cout << "&n: " << &n << endl;
    cout << "&p: " << &p << endl;
    cout << "Sau cap phat => p: " << p << endl;
    cout << "Nhap " << n << " phan tu mang: ";
    for (int i = 0; i < n; i++)
        cin >> p[i];
    for (int i = 0; i < n; i++)
        cout << "&p[" << i << "]": " << p+i << endl;
    return p;
}
int main() {
    int *arr, n;
    cout << "Nhap n: ";
    cin >> n;
    cout << "&n: " << &n << endl;
    cout << "&arr: " << &arr << endl;
    arr = Input(n);
    cout << "\nRa ngoai ham Input: " << endl;
    for (int i = 0; i < n; i++)
        cout << "&arr[" << i << "]": " << arr+i << endl;
    return 0;
}
```

```
Nhap n: 5
&n: 0x61fe0c
&arr: 0x61fe10

Trong ham Input:
&n: 0x61fde0
&p: 0x61fdc0
Sau cap phat => p: 0x731720
Nhap 5 phan tu mang: 1 2 3 4 5
&p[0]: 0x731720
&p[1]: 0x731724
&p[2]: 0x731728
&p[3]: 0x73172c
&p[4]: 0x731730

Ra ngoai ham Input:
&arr[0]: 0x731720
&arr[1]: 0x731724
&arr[2]: 0x731728
&arr[3]: 0x73172c
&arr[4]: 0x731730
```



# Ví dụ: Giải thích cách hoạt động của các hàm bên dưới

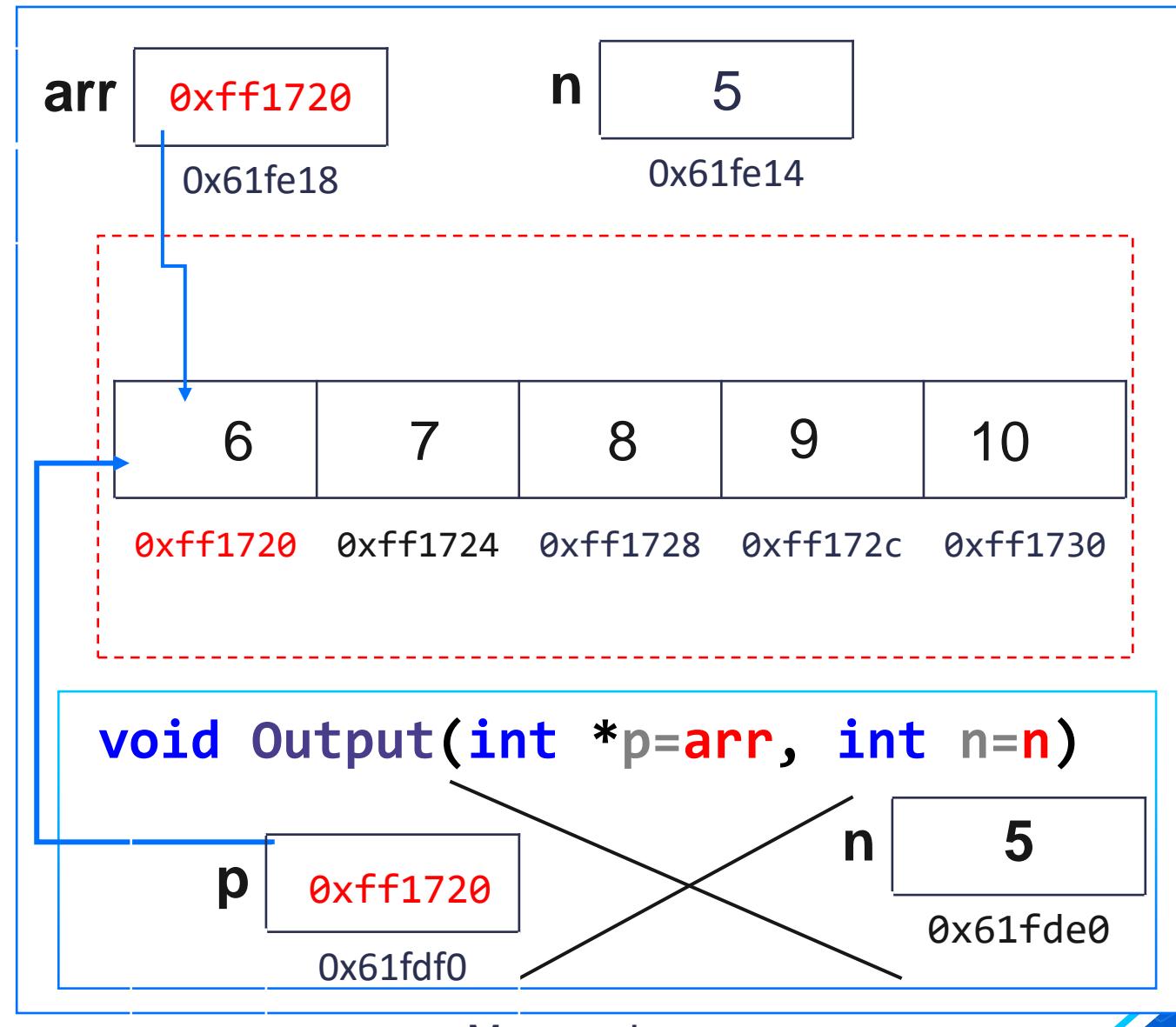
```
#include <iostream>
#include <string.h>
using namespace std;
#define MAXN 300
char *getvalue1(){
    char *temp = "NMLT";
    return temp;
}
char *getvalue2(){
    char temp[] = "CTDL&GT";
    return temp;
}
char *getvalue3() {
    char *temp = new char[MAXN];
    strcpy(temp, "CTDL&GT");
    return temp;
}
```

```
int main() {
    char *s1, *s2, *s3;
    s1 = getvalue1();
    s2 = getvalue2();
    s3 = getvalue3();
    ...
    return 0;
}
```



## 8.15.1 Hàm truyền tham số là con trỏ

```
// Hàm xuất mảng
void Output(int *p, int n) {
    cout << "\n Xuat mang 1 chieu: ";
    for (int i = 0; i < n; i++)
        cout << p[i] << " ";
}
int main() {
    int *arr, n;
    cout << "Nhap n: "; cin >> n;
    arr = Input(n);
    Output(arr, n);
    return 0;
}
```





# Bài tập

- Câu 1: Việc cấp phát động nghĩa là gì?
- Câu 2: Tại sao cần phải giải phóng khối nhớ được cấp phát động?
- Câu 3: Điều gì xảy ra nếu ta nối thêm một số ký tự vào một chuỗi (được cấp phát động trước đó) mà không cấp phát lại bộ nhớ cho nó?
- Câu 4: Ưu điểm của việc sử dụng các hàm thao tác khối nhớ? Ta có thể sử dụng một vòng lặp kết hợp với một câu lệnh gán để khởi tạo hay sao chép các byte nhớ hay không?
- Câu 5: Cho biết sự khác nhau giữa `memcpy` và `memmove`
- Câu 6: Trình bày 2 cách khởi tạo mảng **float data[1000];** với giá trị 0



# Bài tập

- Câu 7: Viết hàm cấp phát một vùng nhớ đủ chứa n số nguyên với n cho trước và trả về địa chỉ vùng nhớ đó.
- Câu 8: Viết hàm sao chép mảng a, số lượng phần tử n cho trước sang mảng b cho trước (kích thước lớn hơn hay bằng n).
- Câu 9: Viết hàm trả về bản sao của một mảng số nguyên a, số lượng phần tử n cho trước.
- Câu 10: Viết hàm trả về mảng đảo của một mảng số nguyên a, số lượng phần tử n cho trước. Yêu cầu không được thay đổi nội dung mảng a.



# Chúc các em học tốt!

