

# **CHƯƠNG 3. CON TRỎ**

**Giáo viên : ThS. Lê Đức Quang**  
**Chuyên môn : KTHT và MTT**  
**Đơn vị : Bộ môn KTHT & MTT**

# NỘI DUNG

---

3.1 Giới thiệu

3.2 Biến con trỏ

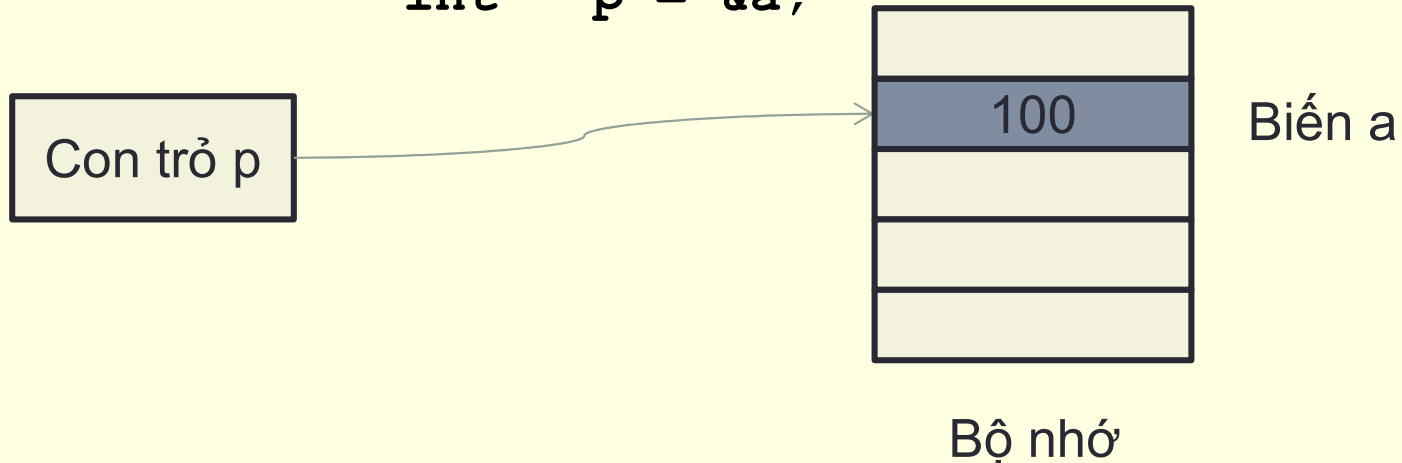
3.3 Con trỏ mảng

3.4 Con trỏ cấu trúc

# 3.1 Giới thiệu

- Con trỏ là gì?
  - Con trỏ thực chất là **địa chỉ bộ nhớ**
  - Nó chỉ ra vị trí lưu giữ một **biến dữ liệu** trong bộ nhớ chương trình

```
int a = 100;  
int * p = &a;
```



Mục đích con trỏ: thao tác và truy nhập biến thông qua địa chỉ của nó

## 3.2 Biến con trỏ

- Biến con trỏ: chứa địa chỉ của đối tượng cần trỏ đến
- Khai báo biến con trỏ
  - Cú pháp: `<kiểu> * <tên biến>;`
  - Ví dụ: khai báo biến con trỏ kiểu `int`, `float`, `double`
    - `int *p; // p trỏ đến một biến kiểu int`
    - `float * pf; // pf trỏ đến một biến kiểu float`
    - `double* plf; // plf trỏ đến một biến kiểu double`
- Khai báo nhiều con trỏ cùng một kiểu
  - `int *p, * q, *u; // p, q, u là 3 con trỏ int`

# Gán địa chỉ cho biến con trỏ

- Cần trỏ đến biến nào, ta gán địa chỉ của biến cho con trỏ

```
int b = 20; int * q;
```

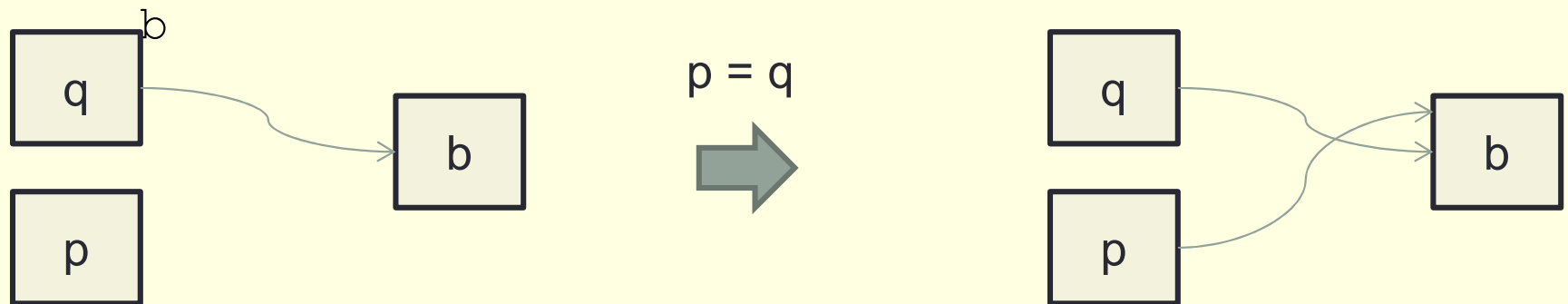
```
q = &b ; // chú ý phải dùng toán tử &
```

```
q = b; // SAI vì q chứa địa chỉ của b chứ  
// không phải giá trị của b
```

- Có thể gán hai biến con trỏ với nhau hoặc gán 1 biểu thức cho con trỏ

```
int * p;
```

```
p = q; // vì q trỏ đến b → cả p và q cùng trỏ đến
```



# Truy nhập giá trị qua biến con trỏ

- Qua biến trỏ, có thể truy nhập giá trị của đối tượng được trỏ

- Cú pháp: **\*<biến trỏ>**

```
float x = 2.0; // Lúc đầu x = 2.0
float *p = &x; // p trỏ đến x
cout<< *p; // kết quả in ra là 2.0
*p = 3.0; // gán 3.0 cho *p
cout<<x; // kết quả in ra là 3.0
           // giá trị x đã thay đổi
```

- **Như vậy**, nếu p trỏ đến x thì \*p có thể thay thế cho biến x trong các phép toán và lệnh

# Kiểu void và con trỏ void

- Kiểu `void` là 1 kiểu dữ liệu đặc biệt, không chứa giá trị cụ thể → gọi là **kiểu trống (hay kiểu rỗng)**
  - KHÔNG dùng để khai báo dữ liệu
- Tuy nhiên, con trỏ kiểu void lại được sử dụng trong trường hợp tổng quát vì có thể nhận giá trị của các kiểu con trỏ khác
  - `int *p; float * q; void * u;`
  - `u = p; u = q; // chấp nhận`
  - `p = q ; // SAI, chương trình dịch cảnh`  
`// báo [warning]`

## 3.3 Con trỏ mảng

- Có thể dùng con trỏ để truy nhập mảng dữ liệu 1 chiều

Cú pháp: `<biến con trỏ> = <tên mảng>;`

Ví dụ:

```
int A[4];
```

```
int *p = A; // Phép gán hợp lệ
```

- Sử dụng con trỏ để truy nhập giá trị phần tử mảng

- `p[0] ⇔ A[0]`

- `p[1] ⇔ A[1]`

- ...

- `p[3] ⇔ A[3]`



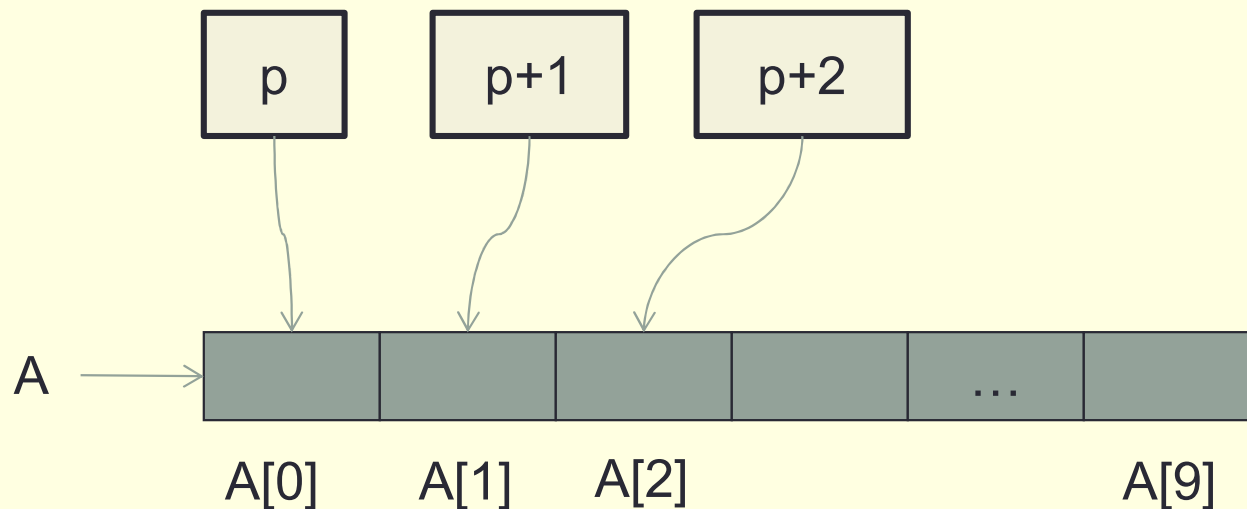
# Phép toán với kiểu con trỏ

- Có thể thực hiện phép + và phép - với biến kiểu con trỏ
- Ví dụ

```
int A[10]; int *p = A; // p trỏ đến A[0]
```

$p + 1$  trỏ đến  $A[1]$

$p + 2$  trỏ đến  $A[2]$



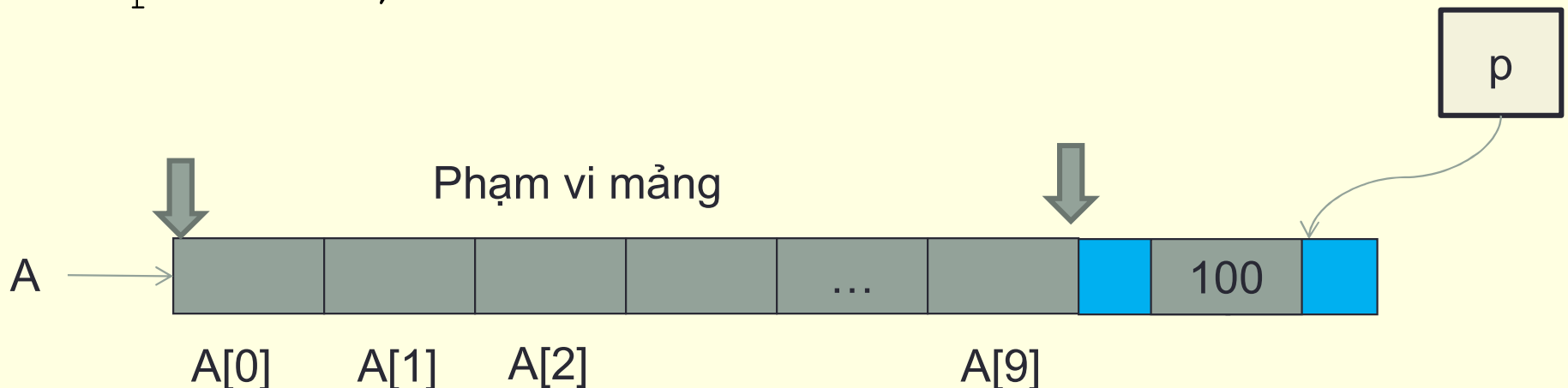
# Chú ý khi sử dụng con trỏ

- Khi con trỏ chưa được khởi tạo, nên gán con trỏ bằng NULL , ví dụ `int *p = NULL;`
- Tránh thay đổi biến con trỏ ra ngoài vùng bộ nhớ được phép

```
int A[10]; int * p = A;
```

```
p += 11; // con trỏ đã vượt ra khỏi mảng A
```

```
*p = 100 ;
```



# Kỹ thuật tạo mảng động

---

- Mảng động là mảng dữ liệu được cấp phát bộ nhớ sau khi chương trình đã chạy
- Trái ngược với mảng tĩnh, kích thước của mảng động không được biết trước

```
int A[10] ; // khai báo mảng tĩnh  
           // số phần tử là 10 đã biết trước
```

- Kỹ thuật tạo mảng động trên C++ dựa trên
  - **Biến con trỏ**
  - Toán tử cấp phát bộ nhớ : `new`
  - Toán tử giải phóng bộ nhớ : `delete`

# Tạo mảng động bằng toán tử new-delete

---

- Cấp phát bộ nhớ bằng từ khóa new:
  - Cú pháp: new <kiểu dữ liệu>[<số phần tử>]
    - Kiểu dữ liệu: tên kiểu dữ liệu của phần tử trong mảng
    - Số phần tử: Số phần tử của mảng
  - Ví dụ:
    - int \* bobby;
    - bobby = new int [5];
- Giải phóng vùng nhớ được cấp phát bằng từ khóa delete
  - Cú pháp: delete <tên mảng động>
  - Ví dụ:
    - delete bobby;

# Tác dụng của mảng động

---

- Sử dụng trong trường hợp số phần tử của mảng không biết trước hoặc không phải hằng số
- Mảng tĩnh chiếm giữ bộ nhớ cho đến khi chương trình kết thúc → không hiệu quả
- Mảng động linh hoạt hơn, khi nào không cần dùng đến có thể thu hồi lại vùng bộ nhớ đã cấp

# Ví dụ: tạo mảng động

---

- Lập trình nhập một số nguyên dương  $n < 20$  từ bàn phím
  - Tạo một mảng động D có n phần tử là số nguyên
  - Nhập giá trị cho mảng
  - In toàn bộ mảng ra màn hình

## 3.4 Con trỏ cấu trúc

- Cấu trúc là 1 kiểu dữ liệu → có kiểu con trỏ tương ứng
- Cú pháp: <tên cấu trúc> \* <tên biến trỏ>;

Ví dụ

```
struct phone_entry
{
    string name; // tên
    string address; // địa chỉ
    long home; // số dt cố định
    long mobile; // số dt di động
};

phone_entry entry = { "Tuan", "So 62 ngo 236", 38529768,
    1689383838 };

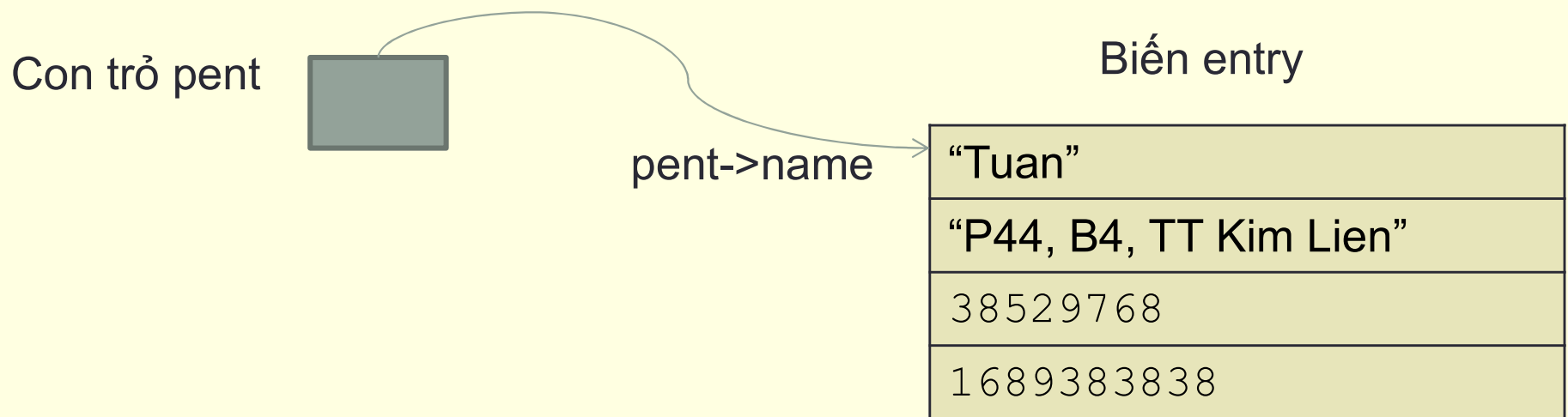
phone_entry* pent = &entry; //gán địa chỉ
```

- Truy nhập trường dữ liệu qua biến con trỏ

**biến\_trỏ->tên\_trường**

VD: pent->address  
pent->home  
pent->mobile

# Minh họa con trỏ cấu trúc



```
phone_entry entry = { "Tuan", "So 62  
Ngo 236", 38529768, 1689383838};  
//gán địa chỉ  
phone_entry* pent = &entry;  
// in ten trong danh ba  
cout<<"Ten:"<<pent->address;
```

Kết quả in ra:

Ten : Tuan



# Tác dụng của con trỏ cấu trúc

---

- Thao tác với con trỏ cấu trúc tương tự như các kiểu con trỏ khác
  - Gán địa chỉ cho con trỏ
  - Cộng / trừ biến con trỏ với số nguyên
  - Khởi tạo giá trị con trỏ = NULL
  - Chú ý không để truy nhập vào vùng nhớ không được phép
- Tác dụng
  - Thao tác với dữ liệu một cách linh hoạt,
  - Sử dụng trong **cấu trúc tự trỏ**

# Mảng con trỏ cấu trúc

- Sử dụng mảng cấu trúc có hạn chế gây lãng phí bộ nhớ khi chưa dùng hết các phần tử
- Kích thước của cấu trúc >> dữ liệu chuẩn → lãng phí nhiều bộ nhớ hơn
  - VD: `phone_entry entrylist[100];`
  - Nếu chỉ dùng 10 phần tử, → lãng phí:  $90 * 56 =$  trong bộ nhớ
- Một giải pháp: sử dụng mảng con trỏ cấu trúc
  - `phone_entry* pentrylist[100];`

# Sử dụng mảng con trỏ cấu trúc

---

- Khi thêm một phần tử : sử dụng **cấp phát bộ nhớ động**

- Thêm phần tử thứ  $i$

- `pentrylist[i] = new phone_entry`

- Truy cập biến cấu trúc thông qua con trỏ

- Nhập dữ liệu cho trường name

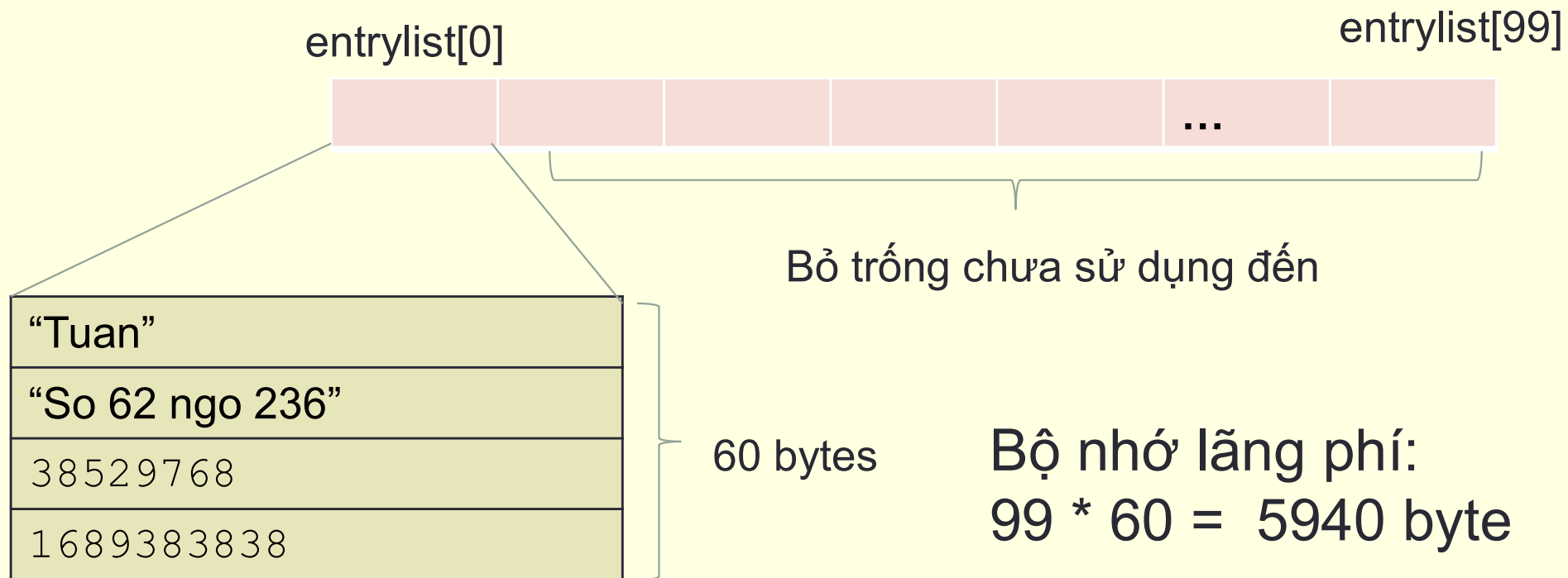
- `gets(pentrylist[i]->name);`

- In dữ liệu cho trường name

- `cout<<"Ten:"<<pentrylist[i]->name;`

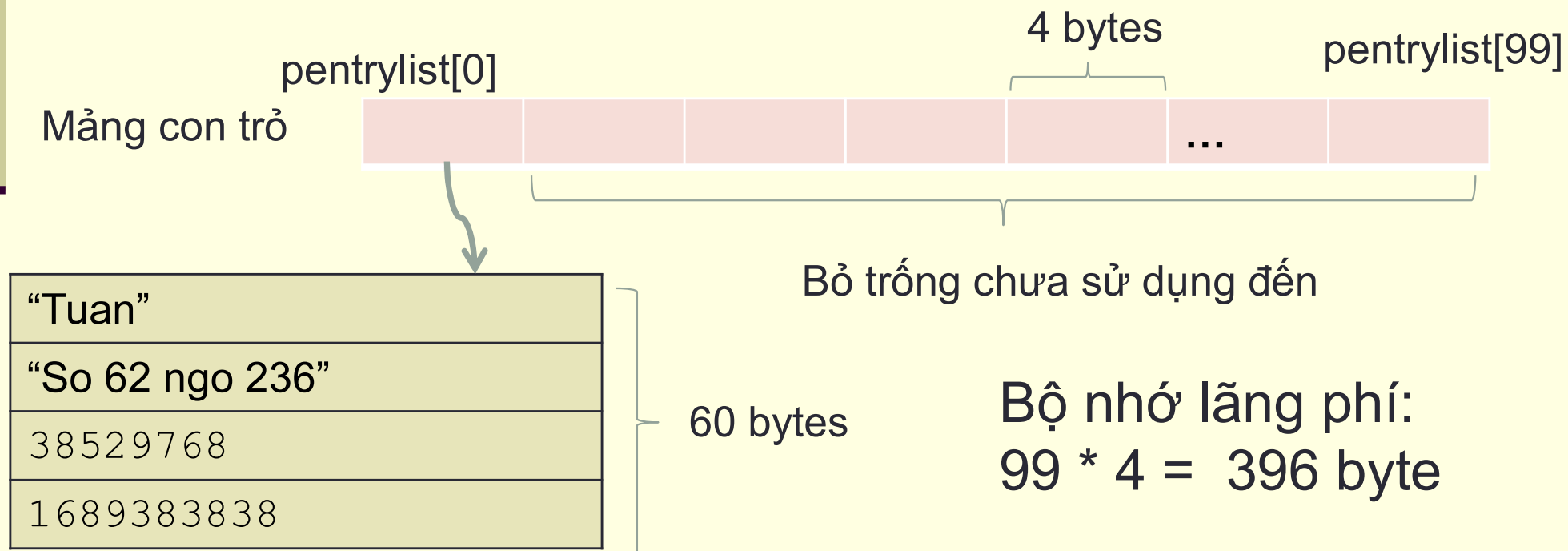
# Minh họa mảng cấu trúc

```
struct phone_entry  entrylist[100]; // 100 phần tử cấu trúc  
// Nhập giá trị cho phần tử entrylist[0]  
...
```



# Minh họa sử dụng mảng con trỏ

```
struct phone_entry * pentrylist[100]; // 100 con trỏ  
//Cấp phát bộ nhớ cho phần tử đầu tiên  
pentrylist[0] = new struct phone_entry;  
gets(pentrylist[0]->name);  
gets(pentrylist[0]->address);  
scanf("%u", &pentrylist[0]->home);  
scanf("%u", &pentrylist[0]->mobile);
```



# Chương trình minh họa

- Lập chương trình quản lý danh bạ
  - Nhập dữ liệu cho từng mục (entry) từ bàn phím
  - In ra toàn bộ các mục theo quy cách

STT	Tên	Địa chỉ	ĐT Nhà	ĐT DD
1	Anh	G3	48473837	1679928737
2	Binh	B4	36903947	904495837

- Tìm một mục trong danh bạ theo tên nhập vào từ bàn phím, nếu tìm thấy thì in ra màn hình, nếu không thì báo là “Không tìm thấy”

## 3.5 Con trỏ lớp

- Tương tự như kiểu cấu trúc, kiểu lớp cũng có con trỏ.

- Cú pháp khai báo:

- `<Lớp> * <con trỏ lớp>`

VD:

```
Point *ptr, p;
```

```
ptr=&p;
```

- Truy cập dữ liệu và hàm thành phần thông qua con trỏ lớp : sử dụng toán tử ->

VD

```
ptr->display();
```

# Cấp phát vùng nhớ động cho con trỏ lớp

- Đối tượng có thể được cấp phát vùng nhớ động bằng từ khóa **new**
  - Khi sử dụng từ khóa **new** để cấp phát vùng nhớ động, chú ý đến các hàm khởi tạo
  - VD:

```
Point *ptr = new Point;  
Point *ptr = new Point();  
Point *ptr = new Point(1, 2);
```
- Giải phóng vùng nhớ động cho đối tượng bằng từ khóa **delete**
  - VD: **delete ptr;**
  - Lưu ý: khi giải phóng vùng nhớ động của đối tượng, đối tượng sẽ bị hủy và gọi đến hàm hủy