Chương 3

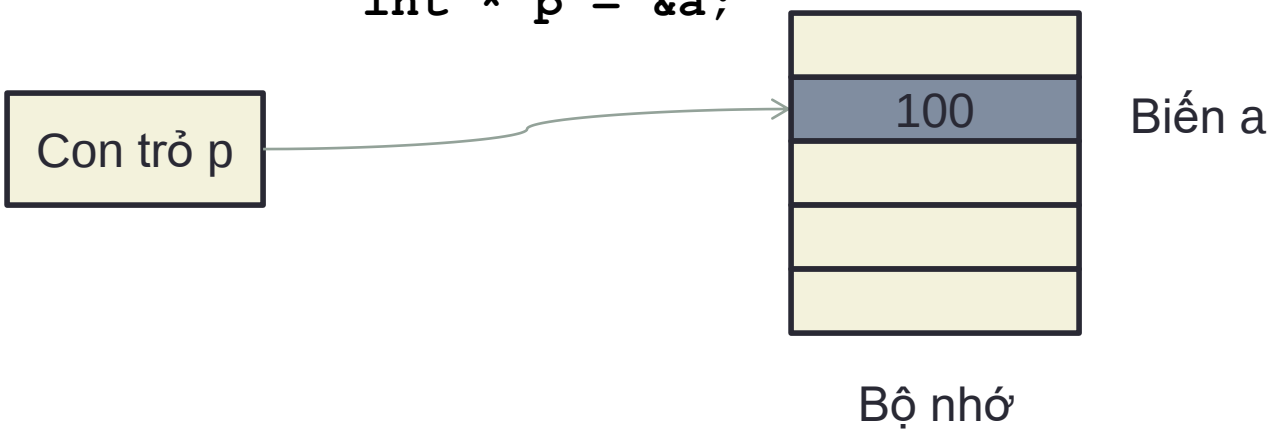
Con trỏ

Giáo viên : ThS. Trần Văn Thọ
Đơn vị : Bộ môn KTHT & MMT

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`

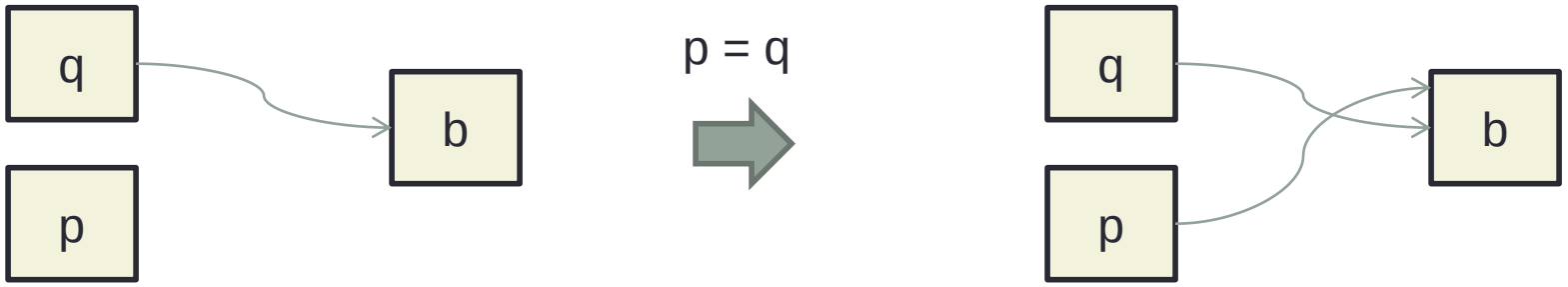
Gán địa chỉ cho biến con trỏ

- 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 b
```



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 báo lỗi`

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]

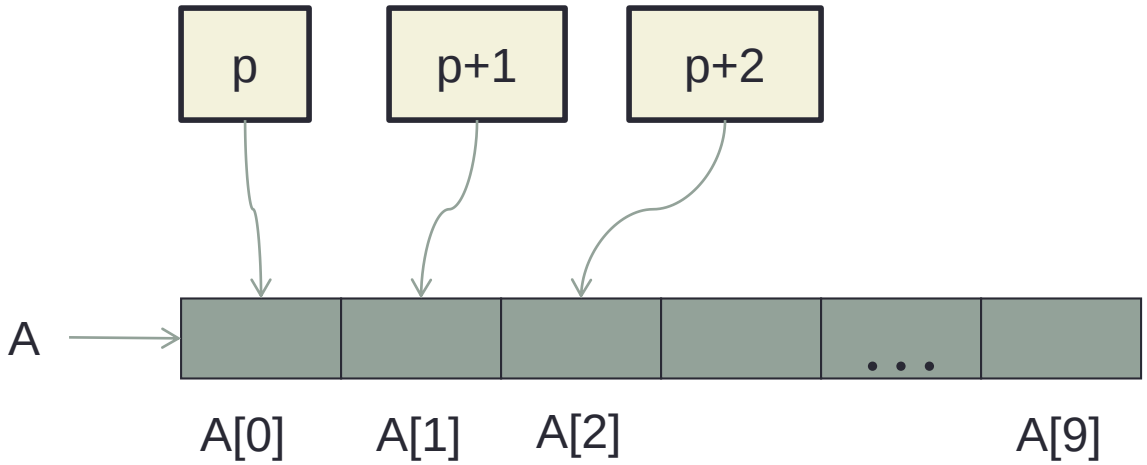
Phép toán với kiểu con trỏ

- 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]

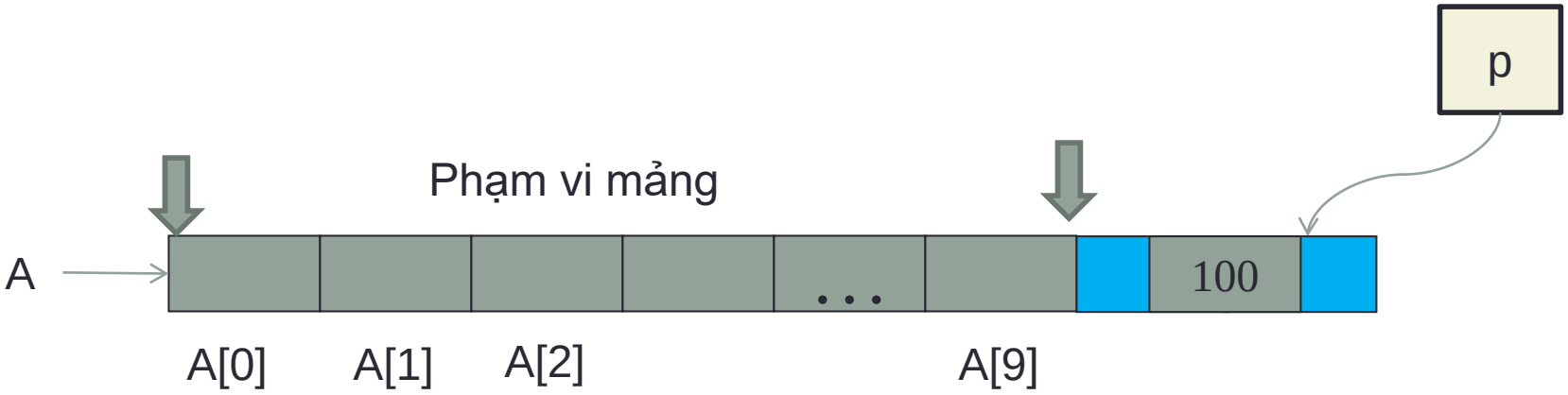
p + 2 trỏ đến A[2]



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>
  - Toán tử [] dùng để báo với hệ điều hành vùng nhớ đã được cấp phát không dùng cho 1 biến đơn.
  - 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
- Vùng nhớ Heap dành cho cấp phát động lớn hơn nhiều vùng nhớ dành cho cấp phát tĩnh (Stack)

Bài tập: tạo mảng động

3.4 Con trỏ cấu trúc

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

Ví dụ:

```
struct  phone_entry
{
   string name;  // tên
   string address; // địa chỉ
    long  home; // số đt cố định
    long  mobile;  // số đt di động
};
struct phone_entry entry = { "Tuan", "So 62 ngo 236",
  38529768, 1689383838 };
struct phone_entry* pent = &entry;//gán địa chỉ
```

- **Truy nhập trường dữ liệu qua biến con trỏ:**

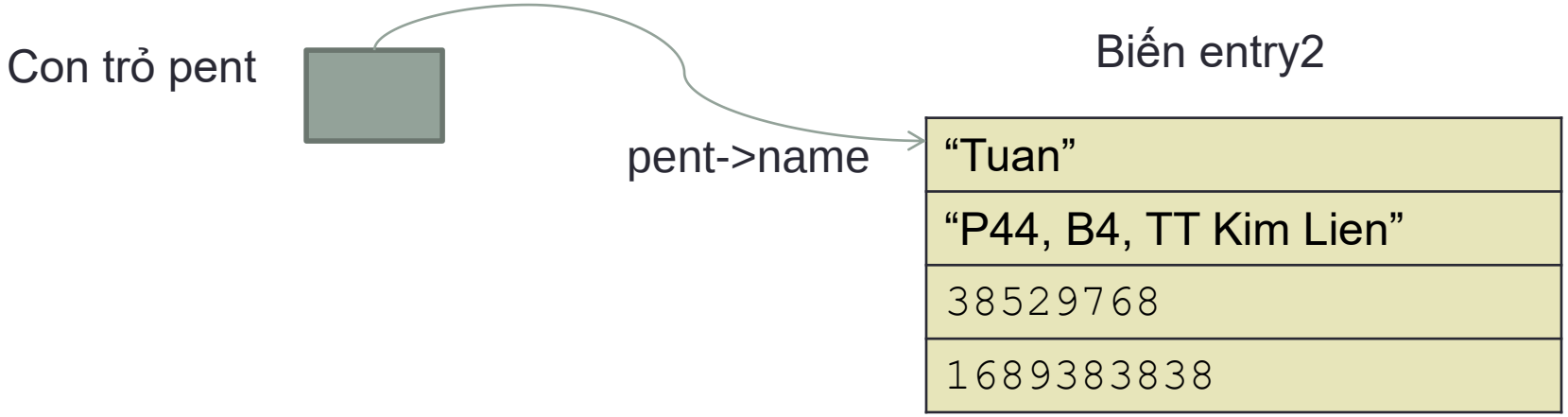
  biến_trỏ->tên_trường

VD: `pent->address`

`pent->home`

`pent->mobile`

Minh họa con trỏ cấu trúc




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

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: `struct phone_entry entrylist[100];`
  - Nếu chỉ dùng 10 phần tử, → lãng phí: 90 * 60 = 5400 trong bộ nhớ
- Một giải pháp: sử dụng mảng con trỏ cấu trúc
  - `struct 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 struct phone_entry`
- Truy cập biến cấu trúc thông qua con trỏ
  - Nhập dữ liệu cho trường name

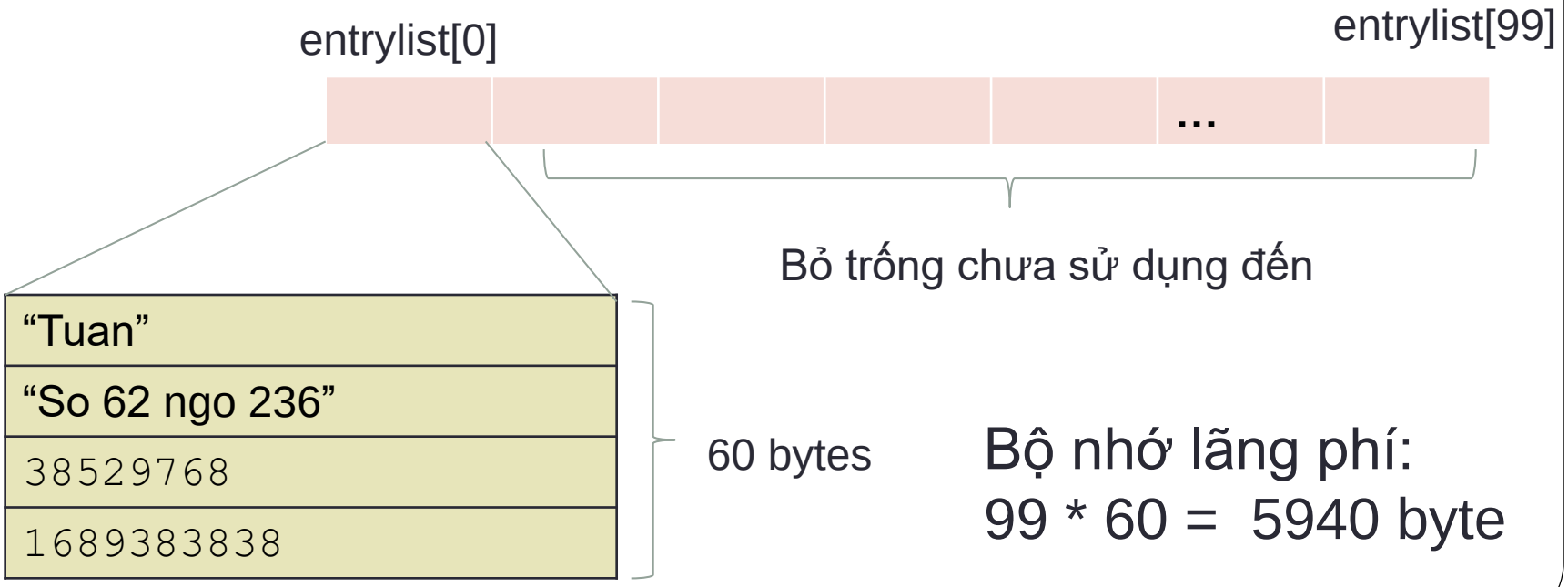
```
getline(cin,pentrylist[i]->name);
```

  - In dữ liệu cho trường name

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

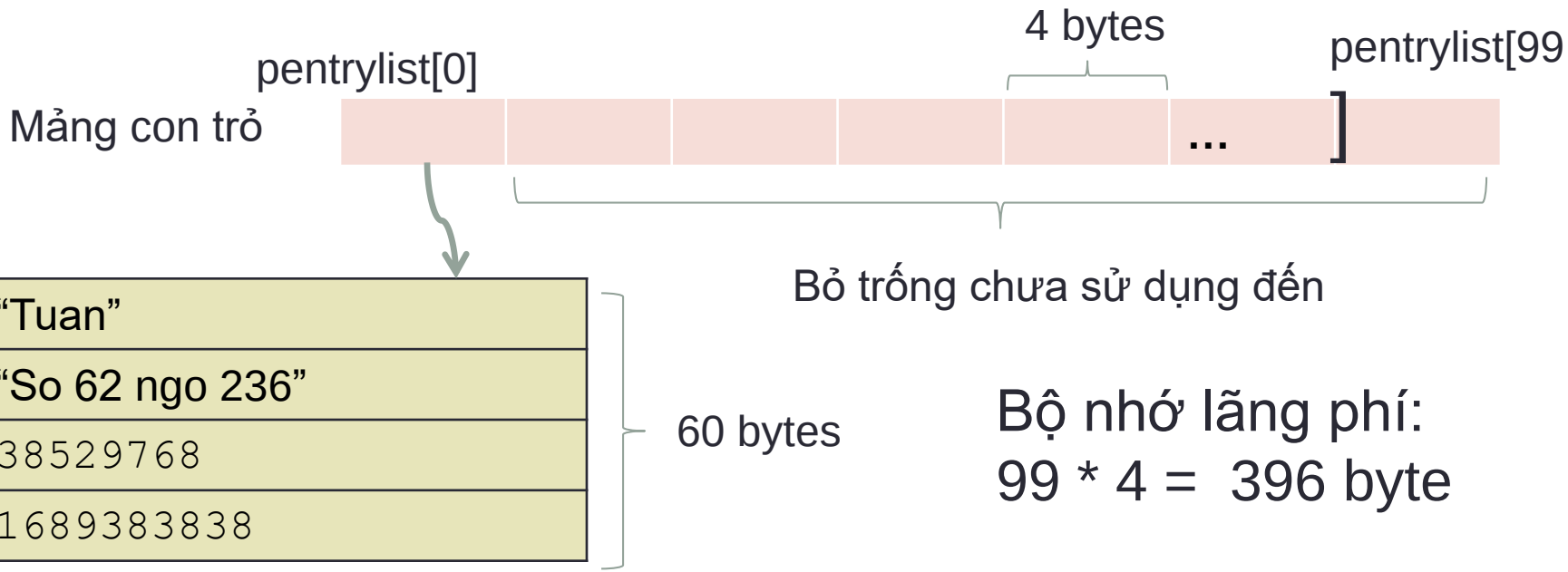
Minh họa mảng cấu trúc

```
struct phone_entry  entrylist[100]; // 100 phần tử cấu trúc
// Nhap gia tri cho phan tu 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;
getline(cin,pentrylist[0]->name);
getline(cin,pentrylist[0]->address);
cin>>pentrylist[0]->home;
cin>>pentrylist[0]->mobile;
```



Chương trình minh họa

- Lập chương trình quản lý danh bạ (Sử dụng mảng con trỏ cấu trúc)
  - 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	ĐTDĐ
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 và lớp

- Chúng ta có thể sử dụng con trỏ là thuộc tính của một lớp hoặc sử dụng lớp làm kiểu con trỏ
- Lớp là 1 kiểu dữ liệu → có kiểu con trỏ tương ứng
- **Cú pháp**: Tên_Lớp * <tên biến trỏ>;

  Ví dụ:

```
class point
{
private:
    int x,y;
public:
    void init();
    void display();
};
point * p = new point;
```

- **Truy nhập các thuộc tính và phương thức biến con trỏ:**

  Cú pháp: biến_trỏ->tên_thuộc_tính

  VD: `p->x`

  `p->y`

  `p->init()`

3.5 Con trỏ và lớp

- Chú ý:
  - Sử dụng con trỏ this để tham chiếu đến chính đối tượng đang gọi.
  - Khi thuộc tính của lớp là con trỏ thì nên định nghĩa lại toán tử gán.
- Ví dụ

```
class dayso
{
    int n;
    int *a;
public:
    void khoitao();
    void seta();
    void xuat();
};
```

```
int main()
{
    dayso a1,a2;
    a1.khoitao();
    a2=a1;
    a1.seta(); //a1 và a2 đều bị thay đổi
}
```