

CHƯƠNG 3

CON TRỎ VÀ BỘ NHỚ

1

GIẢNG VIÊN: NGUYỄN QUỲNH DIỆP

EMAIL: diepnq@tlu.edu.vn

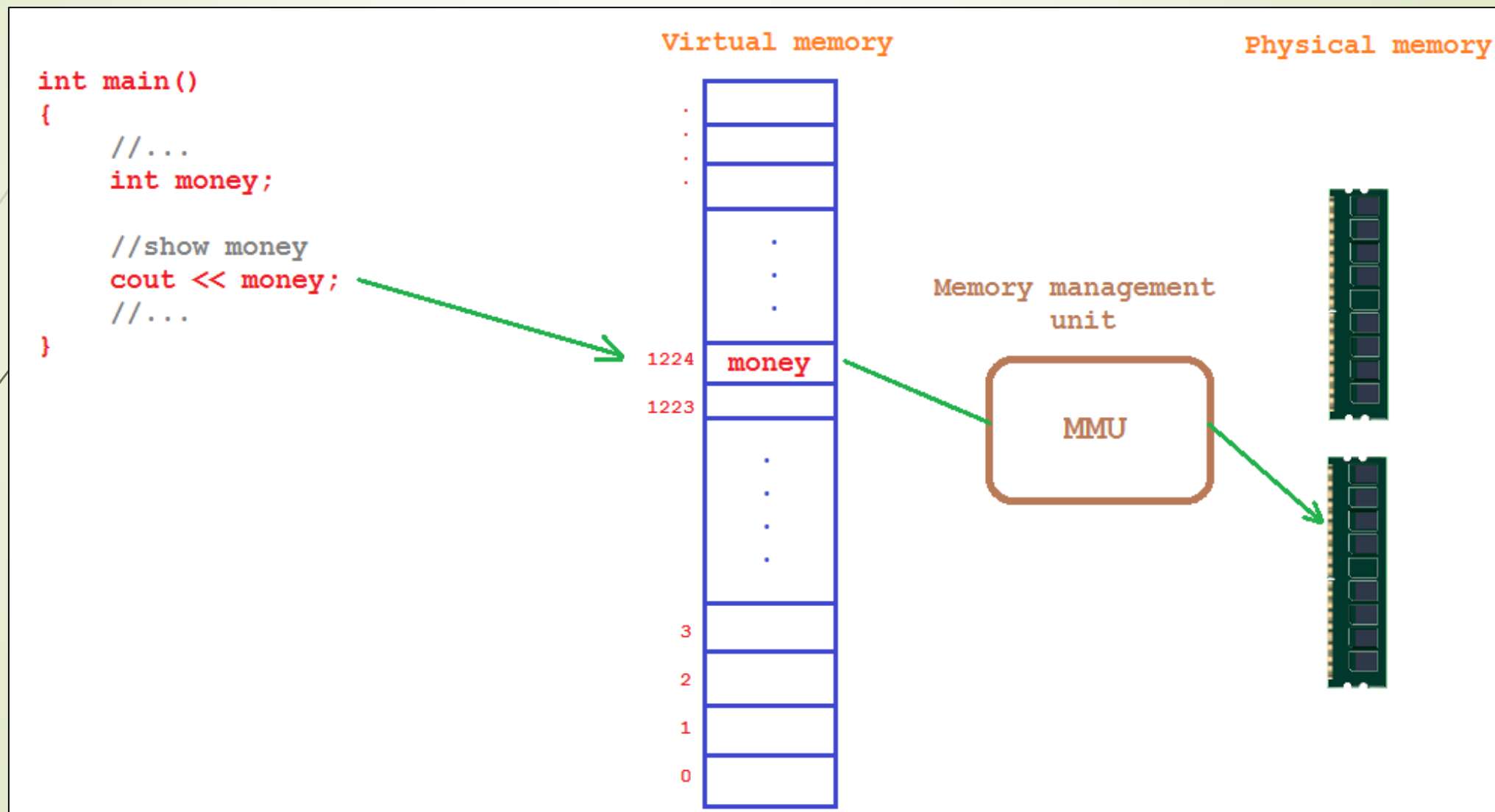
Nội dung

- Bộ nhớ máy tính
- Biến và địa chỉ của biến
- Biến con trỏ, mảng và con trỏ
- Bộ nhớ động
- Mảng động và con trỏ
- Truyền tham số là con trỏ
- Con trỏ hàm

BỘ NHỚ ẢO VÀ BỘ NHỚ VẬT LÝ

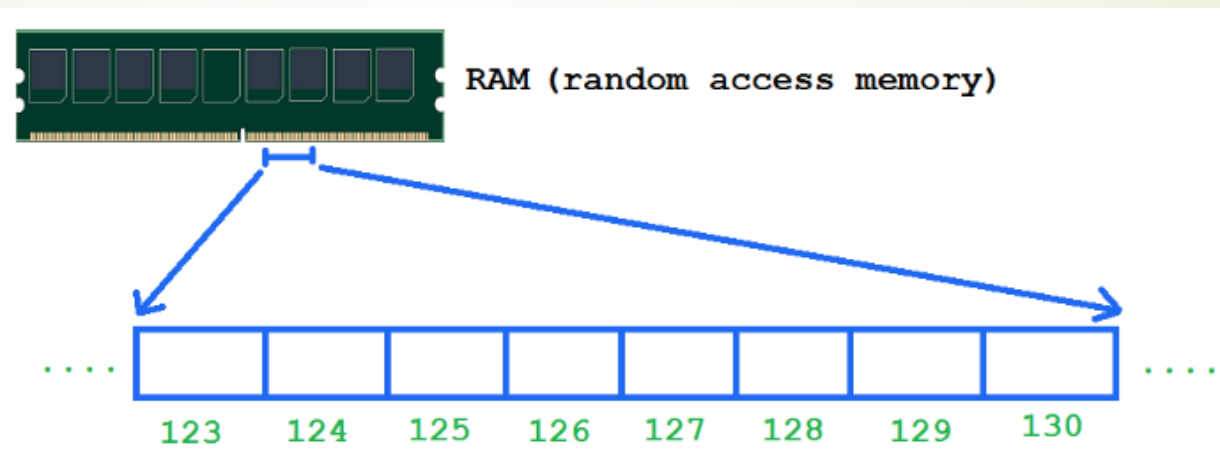
- Việc truy xuất dữ liệu trên bộ nhớ máy tính cần phải:
 - Thông qua một số bước trung gian
 - Không thể trực tiếp truy xuất vào các ô nhớ trên các thiết bị lưu trữ.
- Có thể trỏ đến vùng nhớ ảo (virtual memory) trên máy tính
- Việc truy xuất đến bộ nhớ vật lý (physical memory) từ bộ nhớ ảo phải được thực hiện bởi:
 - Thiết bị phần cứng có tên là **Memory management unit (MMU)** và
 - Một chương trình định vị địa chỉ bộ nhớ gọi là **Virtual address space**.

BỘ NHỚ ẢO VÀ BỘ NHỚ VẬT LÝ



BỘ NHỚ ẢO VÀ BỘ NHỚ VẬT LÝ

- RAM hay các thiết bị cung cấp bộ nhớ tạm thời khác đều được tạo nên bởi các ô nhớ liên tiếp nhau.
- Mỗi ô nhớ đều có 1 số thứ tự đại diện cho vị trí của ô nhớ đó trong thiết bị lưu trữ, đó là địa chỉ của ô nhớ.
- Địa chỉ ô nhớ được đánh số từ 0, địa chỉ cuối cùng phụ thuộc vào số nhớ trên thiết bị.
- Địa chỉ của biến được định dạng theo hệ cơ số 16
- 1 ô nhớ trong thiết bị lưu trữ như là một cái nhà trên con đường, để xác định được vị trí của cái nhà, chúng ta cần biết địa chỉ của nhà đó.



CÁCH LÀM VIỆC CỦA BỘ NHỚ

- Khởi tạo chương trình: vùng nhớ được cấp phát khi khởi động chương trình
- Nếu khai báo:
 - 1 biến int \Rightarrow chiếm 4 byte
 - 1 biến char \Rightarrow chiếm 1 byte
 - 1 biến double \Rightarrow 8 byte
 - 1 biến float \Rightarrow 4 byte
 - ...
 - 1 biến mảng double 1 tỷ phần tử \Rightarrow tràn bộ nhớ





Giải
pháp?



Con trỏ

A 3D rendered yellow figure stands behind a large, light-yellow rectangular sign. The figure's hands are visible, holding the top and bottom edges of the sign. The sign has a thin yellow border and a subtle texture. The background is a plain, light yellow gradient. On the left side, there are several thin, dark, curved lines that appear to be part of a larger graphic element.

Con trỏ

CON TRỎ HOẠT ĐỘNG NHƯ THẾ NÀO?

- Khởi tạo chương trình
- Nếu khai báo:
 - 1 biến con trỏ kiểu bất kỳ

=> 4 byte hoặc 8 byte tùy theo HĐH 32 bit hoặc 64 bit

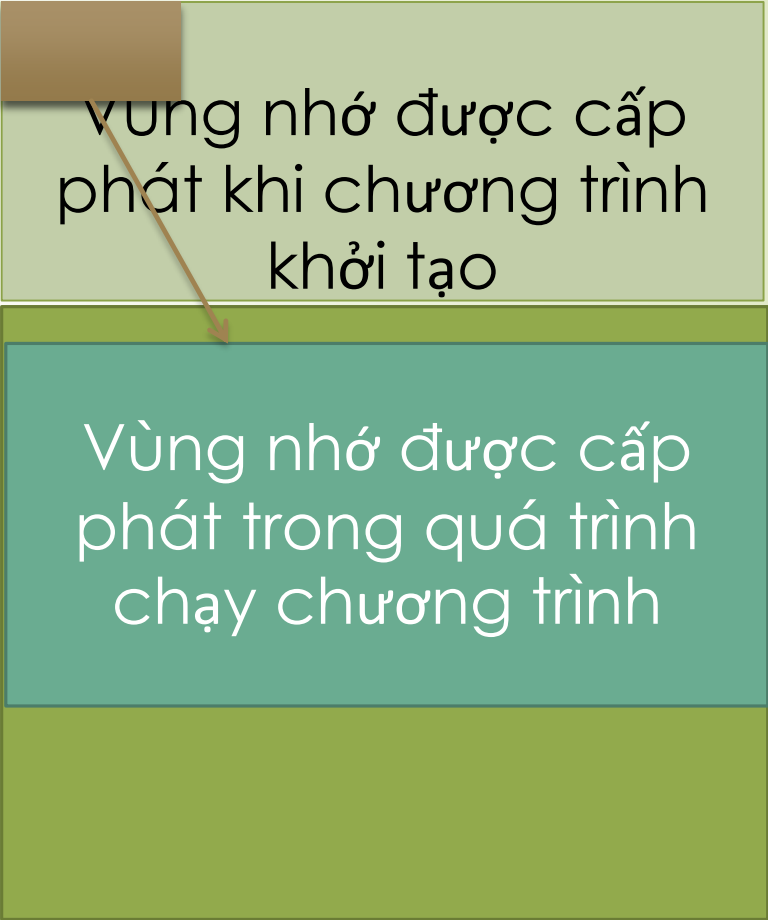
- Khi dùng: Con trỏ sẽ được gán trỏ đến 1 vùng địa chỉ để làm việc



vùng nhớ được cấp phát khi khởi
động chương trình

CON TRỎ HOẠT ĐỘNG NHƯ THẾ NÀO?

- Khi cần bộ nhớ lớn hơn để làm việc thì sẽ được cấp phát động trong quá trình chạy, vùng nhớ được cấp phát động sẽ nằm ngoài vùng nhớ mặc định mà chương trình có được khi khởi tạo



Vùng nhớ được cấp phát khi chương trình khởi tạo

The diagram shows a vertical stack of three rectangular boxes. The top box is light green and contains the text 'Vùng nhớ được cấp phát khi chương trình khởi tạo'. The middle box is teal and contains the text 'Vùng nhớ được cấp phát trong quá trình chạy chương trình'. The bottom box is olive green and is empty. A brown tab is attached to the top-left corner of the top box. A brown arrow points from the bottom-right corner of the top box to the top-left corner of the middle box.

Vùng nhớ được cấp phát trong quá trình chạy chương trình

KHÁI NIỆM

- Con trỏ là biến chứa địa chỉ của một vùng nhớ, nên khi gán giá trị cho con trỏ thì giá trị đó phải là 1 địa chỉ
- Mục đích: nhằm sử dụng bộ nhớ một cách linh hoạt và tiết kiệm

KHAI BÁO

- Tương tự như khai báo các biến thông thường
- Có dấu ***** trước tên biến, gọi là toán tử trở.
- Cú pháp: **<tên kiểu> *<tên biến con trở>**
- Mục đích: Tạo ra con trở trở tới kiểu đó
- Ví dụ: `int *p`: con trở p trở tới biến kiểu `int`
`double *t`: con trở t trở tới biến kiểu `double`
- Truy xuất địa chỉ của biến con trở: `<tên biến con trở>`
- Truy xuất đến giá trị của biến con trở: `*<tên biến con trở>`

TOÁN TỬ &

- Toán tử **&** của biến: dùng để xác định (lấy) địa chỉ của 1 biến đó.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x;
6      cout <<"Dia chi cua bien la: "<<&x;
7  }
```

C:\DIEP-DATA\OneDrive - Thuyloi University\LT NANGCAO\contro.exe

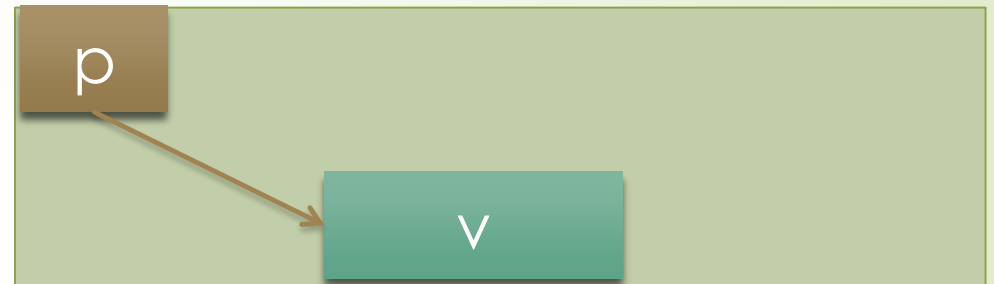
Dia chi cua bien la: 0x22fe4c

Process exited after 0.02228 seconds with return value 0
Press any key to continue . . .

TOÁN TỬ &

- ➡ Thiết lập biến con trỏ trỏ tới một địa chỉ:

```
int *p;  
int v;  
p = &v; // p trỏ tới v
```



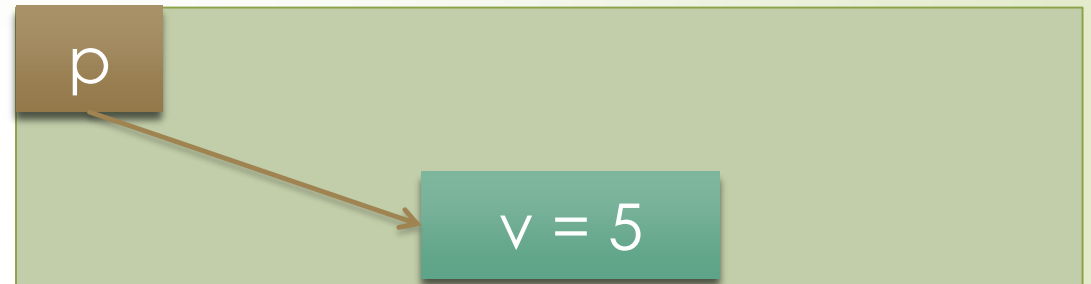
Ý nghĩa: Thiết lập biến con trỏ p, trỏ tới biến nguyên v

- ➡ Cách đọc: p trỏ tới v
hoặc p bằng địa chỉ của v

TOÁN TỬ *

- ➡ Lấy giá trị tại vị trí con trỏ đang trỏ tới

```
int *p;  
int v;  
p = &v; // p trỏ tới v  
v = 5;  
cout << *p; // => 5
```

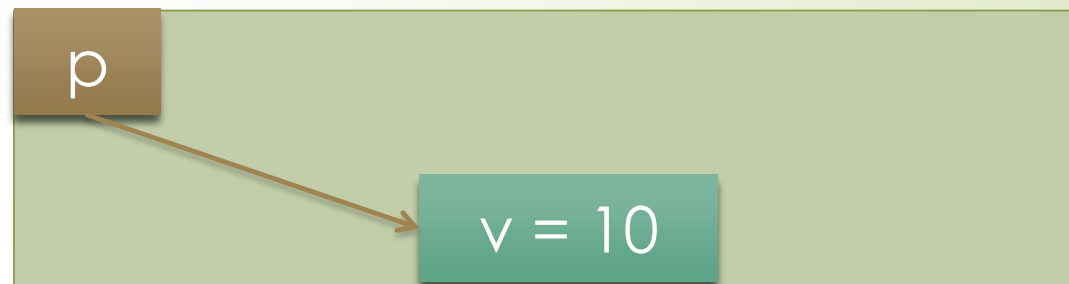


- ➡ Toán tử * dùng để khử tham chiếu.

TOÁN TỬ *

- ➡ Gán giá trị cho vị trí con trỏ đang trỏ tới

```
int *p;  
int v;  
p = &v; // p trỏ tới v  
*p = 10; // => v = 10
```



- ➡ Toán tử * dùng để khử tham chiếu.

VÍ DỤ 1

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int x=3;
6      cout <<"Gia tri cua bien x = "<<x;
7      cout <<"\nDia chi cua bien la: "<<&x;
8      int *p;
9      p=&x;
10     cout <<"\nGia tri cua con tro p la: "<<p;
11     cout <<"\nGia tri cua bien duoc tro boi p la: "<<*p;
12 }
13
```

C:\DIEP-DATA\OneDrive - Thuyloi University\LT NANGCAO\contro.exe

```
Gia tri cua bien x = 3
Dia chi cua bien la: 0x22fe34
Gia tri cua con tro p la: 0x22fe34
Gia tri cua bien duoc tro boi p la: 3
```

```
-----
Process exited after 0.0163 seconds with return value 0
Press any key to continue . . .
```

VÍ DỤ 2

➡ Lệnh:

```
v=0;
```

```
p=&v;
```

```
*p = 10
```

```
cout<< v<< endl;
```

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

➡ Kết quả:

10

10

PHÉP GÁN CON TRỎ

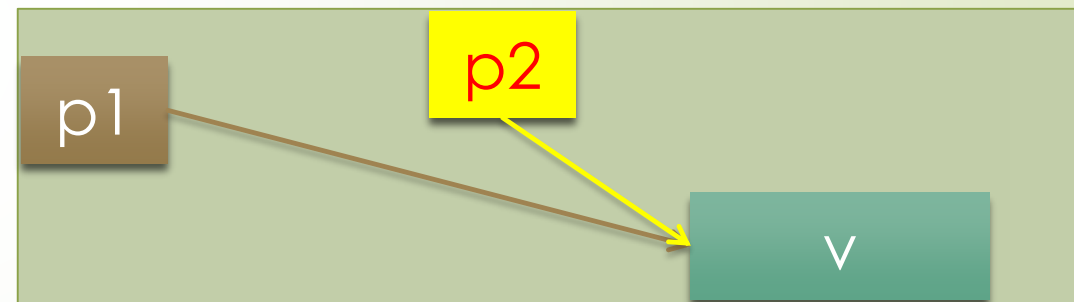
➤ Cú pháp:

<con trỏ 1> = <con trỏ 2>

➤ Tác dụng: thiết lập 2 con trỏ cùng trỏ tới một địa chỉ.

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

```
p1 = &v; // p1 trỏ tới v
```



➤ Lệnh gán: $p2 = p1$ là làm cho $p2$ trỏ tới nơi mà $p1$ đang trỏ đến.

PHÉP GÁN GIÁ TRỊ CON TRỎ

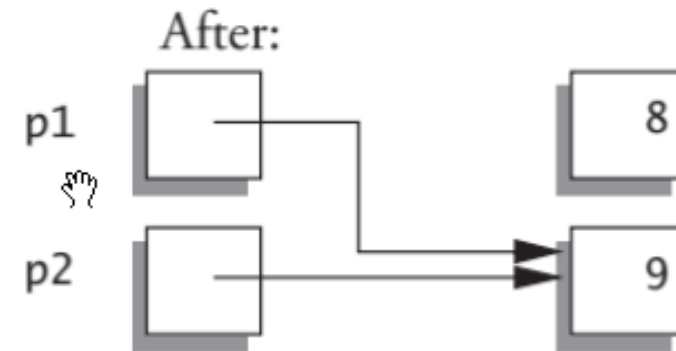
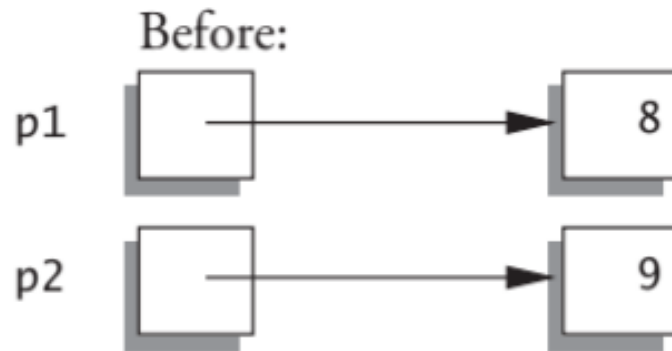
➤ Cú pháp:

***<con trỏ 1> = *<con trỏ 2>**

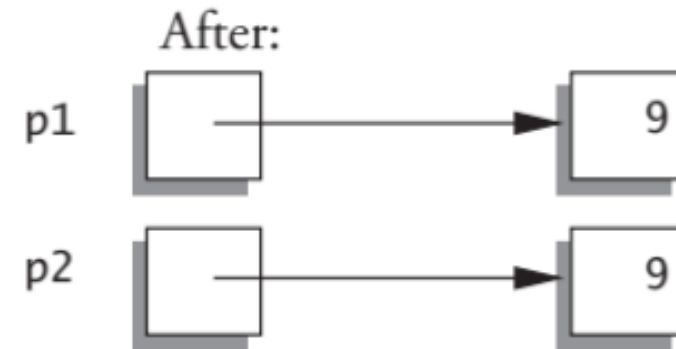
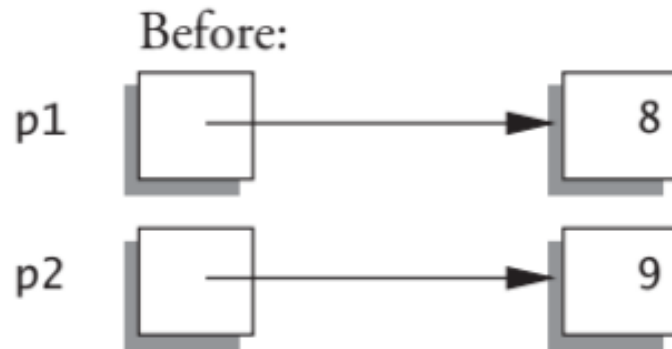
➤ Tác dụng: Thiết lập 2 con trỏ có cùng giá trị, gán giá trị của con trỏ 1 bằng giá trị của con trỏ 2

VÍ DỤ VỀ PHÉP GÁN CON TRỎ

`p1 = p2;`



`*p1 = *p2;`



CẤP PHÁT BỘ NHỚ

➤ Cấp phát bộ nhớ tĩnh (Static memory allocation):

- Xảy ra trên các biến tĩnh và biến toàn cục.
- Vùng nhớ của các loại biến này được cấp phát một lần khi chương trình bắt đầu chạy và vẫn tồn tại trong suốt thời gian tồn tại của chương trình.
- Kích thước của biến/mảng phải được biết tại thời điểm biên dịch chương trình.

➤ Cấp phát bộ nhớ tự động (Automatic memory allocation):

- Xảy ra trên các tham số hàm và biến cục bộ.
- Vùng nhớ của các loại biến này được cấp phát khi chương trình đi vào khối lệnh và được giải phóng khi khối lệnh bị thoát.
- Kích thước của biến/mảng phải được biết tại thời điểm biên dịch chương trình.

CẤP PHÁT BỘ NHỚ

➤ Cấp phát bộ nhớ động (Dynamic memory allocation)

- Dùng để cấp phát bộ nhớ tại thời điểm run-time.
- Tại thời điểm này, không thể tạo ra tên biến mới, mà chỉ có thể tạo ra vùng nhớ mới.
- Để kiểm soát được những vùng nhớ được cấp phát bằng kỹ thuật **Dynamic memory allocation** là sử dụng con trỏ lưu trữ địa chỉ đầu tiên của vùng nhớ được cấp phát, thông qua con trỏ để quản lý vùng nhớ trên **Heap**.

➤ Cú pháp cấp phát vùng nhớ động:

<con trỏ> = new <kiểu dữ liệu>

➤ Cú pháp hủy cấp phát vùng nhớ động:

Delete <con trỏ>

CẤP PHÁT VÙNG NHỚ ĐỘNG CHO CON TRỎ

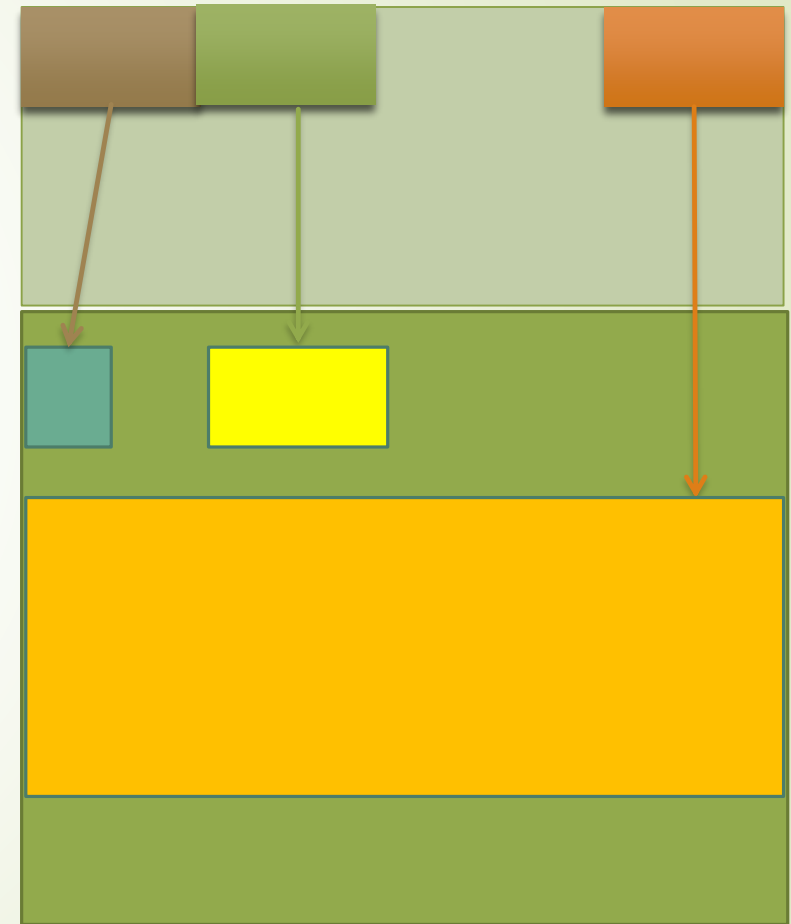
- Vì con trỏ được dùng để linh hoạt trong quá trình chạy CT nên khi cần mới cấp phát bộ nhớ cho nó
- Sử dụng toán tử **new**

```
char *p1 = new char;
```

```
int *p2 = new int;
```

```
double *p3 = new double[1000];
```

- Nếu việc cấp phát không thành công thì con trỏ sẽ có giá trị **NULL**



HỦY VÙNG NHỚ ĐỘNG CỦA CON TRỎ

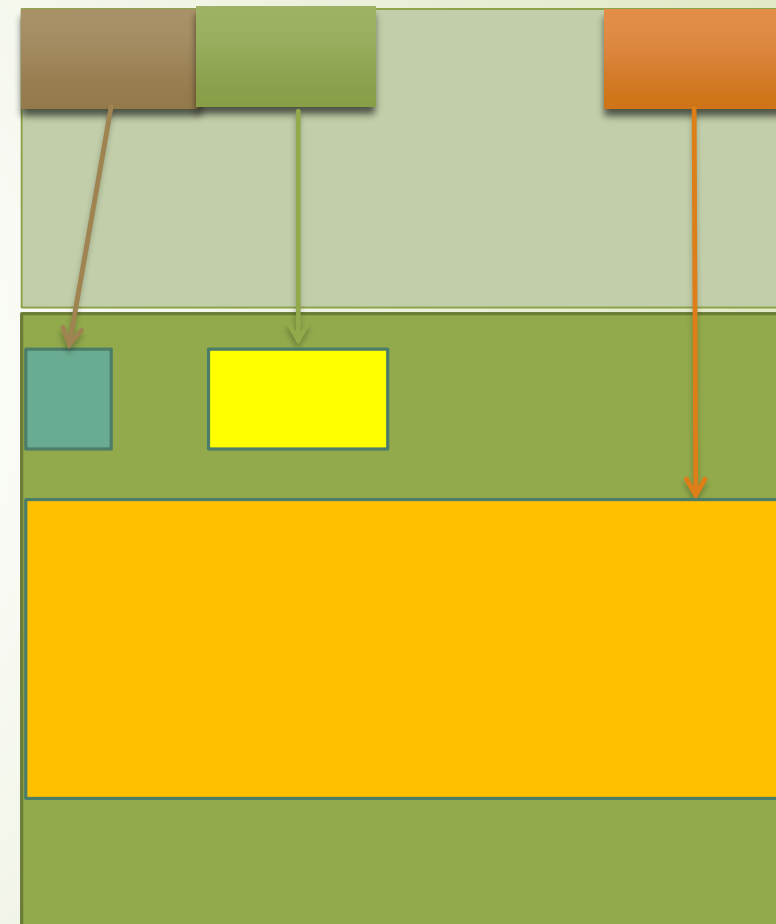
- Sử dụng toán tử **delete**

```
delete p1;
```

```
delete p2;
```

```
delete p3;
```

- Sau khi gọi những lệnh trên, con trỏ vẫn trỏ tới vùng ô nhớ đã bị xóa



HỦY VÙNG NHỚ ĐỘNG CỦA CON TRỎ

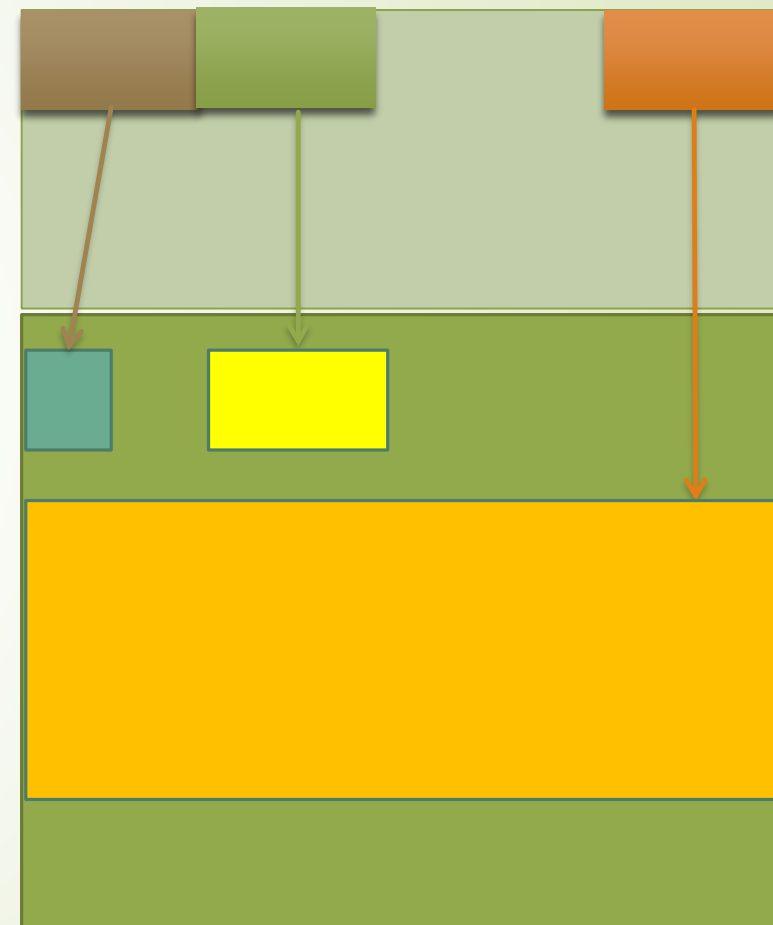
- Sử dụng toán tử **delete**

```
delete p1; p1 = NULL;
```

```
delete p2; p2 = NULL;
```

```
delete p3; p3 = NULL;
```

- Phải gán con trỏ = NULL để đảm bảo con trỏ không trỏ tới vùng nhớ đã bị xóa
- Việc gán giá trị NULL sẽ tránh tình trạng con trỏ trỏ đến 1 vùng nhớ không xác định



VÍ DỤ CẤP PHÁT CON TRỎ

```
1 //Chương trình minh họa con trỏ và biến động.
2 #include <iostream>
3 using std::cout;
4 using std::endl;
5 int main()
6 {
7     int *p1, *p2;
8     p1 = new int;
9     *p1 = 42;
10    p2 = p1;
11    cout << "*p1 == " << *p1 << endl;
12    cout << "*p2 == " << *p2 << endl;
13    *p2 = 53;
14    cout << "*p1 == " << *p1 << endl;
15    cout << "*p2 == " << *p2 << endl;
16    p1 = new int;
17    *p1 = 88;
18    cout << "*p1 == " << *p1 << endl;
19    cout << "*p2 == " << *p2 << endl;
20    cout << "Mong ban hieu duoc vi du nay!\n";
21    return 0;
22 }
```


VÍ DỤ CẤP PHÁT CON TRỞ

➡ Kết quả chạy chương trình

```
*p1 == 42  
*p2 == 42  
*p1 == 53  
*p2 == 53  
*p1 == 88  
*p2 == 53  
Mong ban hieu duoc vi du nay!
```

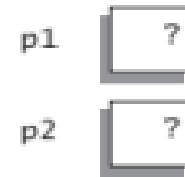

VÍ DỤ CẤP PHÁT CON TRỞ

```

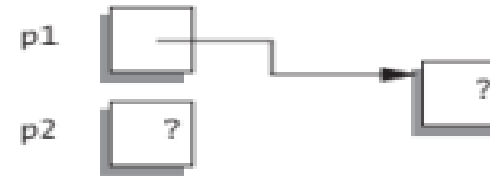
1 //Chương trình minh họa con trỏ và biến động.
2 #include <iostream>
3 using std::cout;
4 using std::endl;
5 int main()
6 {
7     int *p1, *p2;
8     p1 = new int;
9     *p1 = 42;
10    p2 = p1;
11    cout << "*p1 == " << *p1 << endl;
12    cout << "*p2 == " << *p2 << endl;
13    *p2 = 53;
14    cout << "*p1 == " << *p1 << endl;
15    cout << "*p2 == " << *p2 << endl;
16    p1 = new int;
17    *p1 = 88;
18    cout << "*p1 == " << *p1 << endl;
19    cout << "*p2 == " << *p2 << endl;
20    cout << "Mong ban hieu duoc vi du nay!\n";
21    return 0;
22 }

```

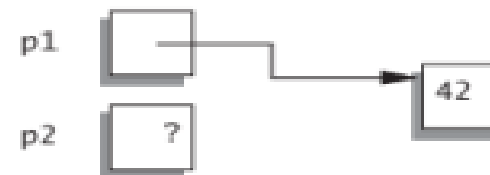
(a)
int *p1, *p2;



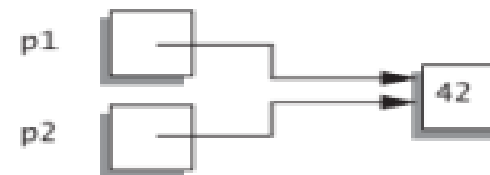
(b)
p1 = new int;



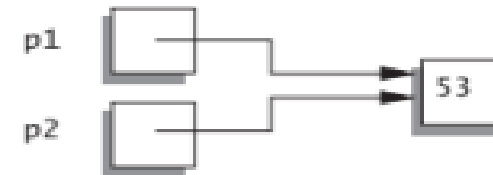
(c)
*p1 = 42;



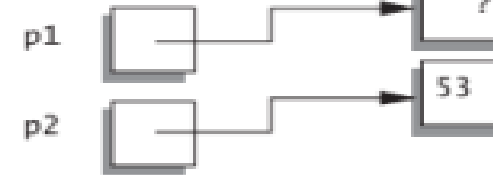
(d)
p2 = p1;



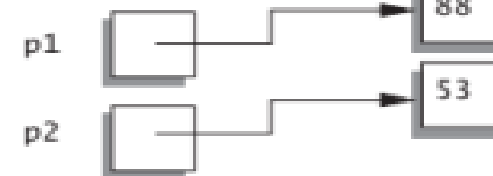
(e)
*p2 = 53;



(f)
p1 = new int;



(g)
*p1 = 88;

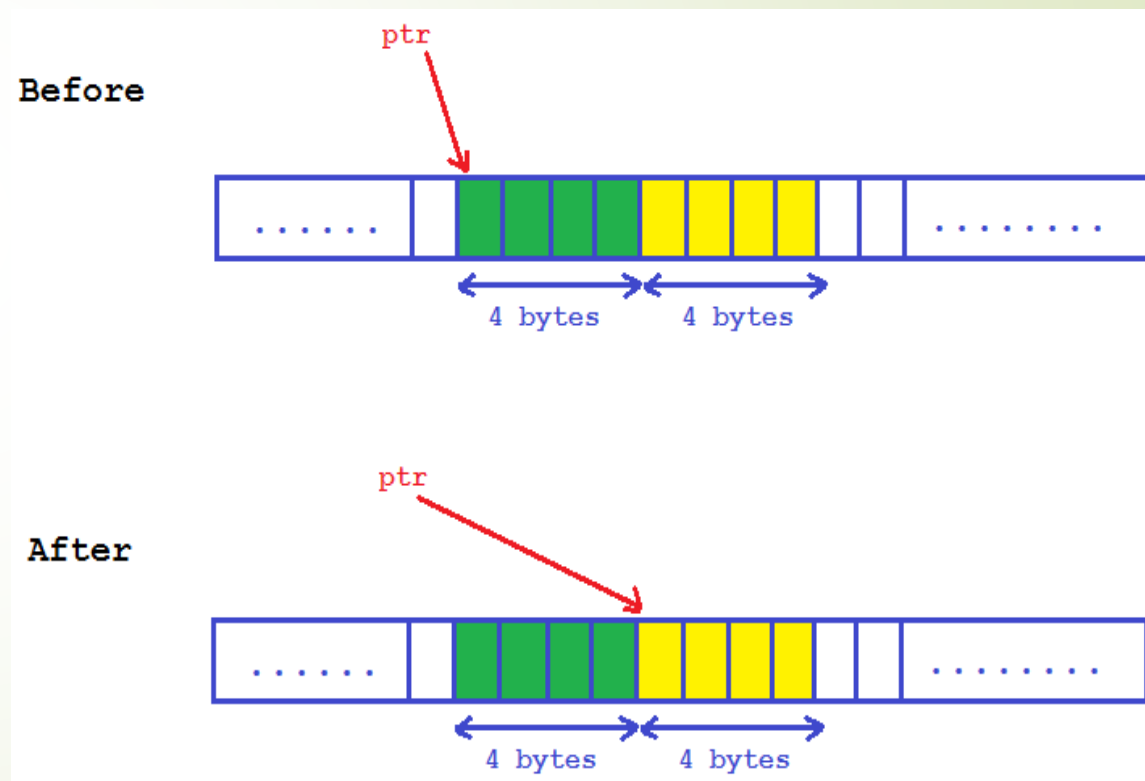


CÁC PHÉP TOÁN TRÊN CON TRỎ

4 toán tử toán học có thể sử dụng cho con trỏ: ++, --, +, và -

➤ Increment operator (++):

- Cú pháp: **<con trỏ> ++**
- Tác dụng: dùng để tăng giá trị bên trong vùng nhớ của biến lên 1 đơn vị.
- Làm con trỏ trỏ đến địa chỉ tiếp theo trên bộ nhớ ảo.
- Khoảng cách của 2 địa chỉ này đúng bằng kích thước của kiểu dữ liệu được khai báo cho con trỏ



CÁC PHÉP TOÁN TRÊN CON TRỎ

➡ Ví dụ:

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main() {
6     int value = 0;
7     int *ptr = &value;
8
9     cout << "Before increased: " << ptr << endl;
10
11     ptr++;
12     cout << " After increased: " << ptr << endl;
13     return 0;
14 }
```

Before increased: 0x6ffe14
After increased: 0x6ffe18

CÁC PHÉP TOÁN TRÊN CON TRỎ

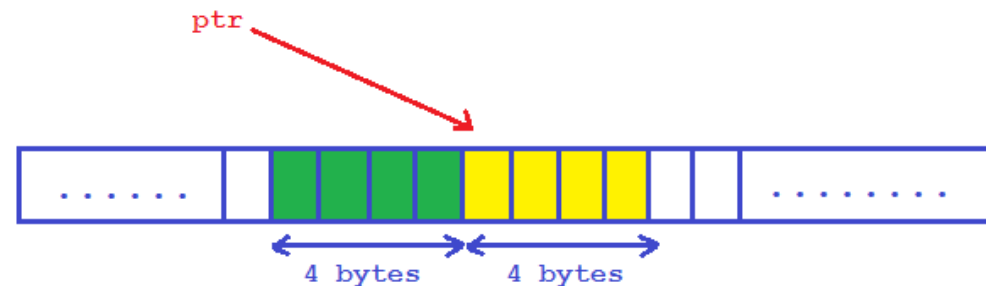
► Decrement operator (--):

► Cú pháp: **<con trỏ> --**

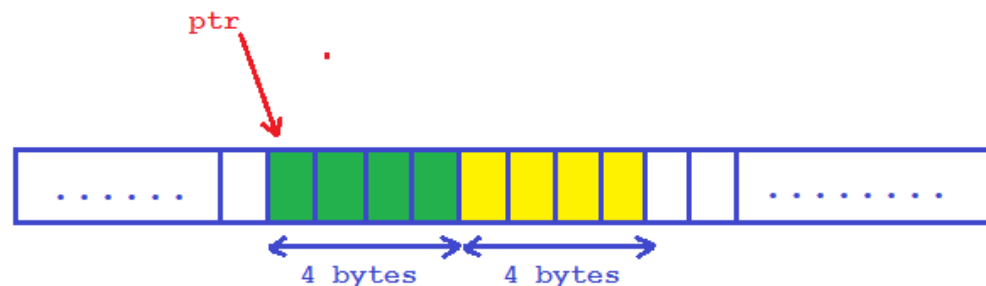
► Tác dụng: dùng để giảm giá trị bên trong vùng nhớ của biến thông thường đi 1 đơn vị.

► Làm thay đổi địa chỉ của con trỏ đang trỏ đến, giá trị địa chỉ mới sẽ bằng giá trị địa chỉ cũ trừ đi kích thước của kiểu dữ liệu mà con trỏ đang trỏ đến..

Before



After



CÁC PHÉP TOÁN TRÊN CON TRỎ

➡ Ví dụ:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6      int value = 0;
7      int *ptr = &value;
8
9      cout << "Before decreased: " << ptr << endl;
10
11     ptr--;
12     cout << " After decreased: " << ptr << endl;
13     return 0;
14 }
```

Before decreased: 0x6ffe14
After decreased: 0x6ffe10

CÁC PHÉP TOÁN TRÊN CON TRỎ

➤ Addition operator (+):

➤ Cú pháp:

<con trỏ> + n

➤ Tác dụng: trỏ đến vùng nhớ bất kỳ phía sau địa chỉ mà con trỏ đang nắm giữ

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6      int value = 0;
7      int *ptr = &value;
8
9      cout << ptr << endl;
10
11     ptr = ptr + 5;
12     cout << ptr << endl;
13     return 0;
14 }
```

```
0x6ffe14
0x6ffe28
```


CÁC PHÉP TOÁN TRÊN CON TRỎ

➤ Subtraction operator (-):

➤ Cú pháp:

<con trỏ> - n

➤ Trỏ đến vùng nhớ bất kỳ phía trước địa chỉ mà con trỏ đang nắm giữ

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6      int value = 0;
7      int *ptr = &value;
8
9      cout << ptr << endl;
10
11     ptr = ptr - 5;
12     cout << ptr << endl;
13     return 0;
14 }
```

0x6ffe14


0x6ffe00

ĐỊNH NGHĨA KIỂU CỦA CON TRỎ

- Có thể đặt tên các kiểu con trỏ
- Để có thể khai báo con trỏ giống như những biến khác
 - Loại bỏ * trong khai báo con trỏ
- Khai báo: Sử dụng từ khóa **typedef**

```
typedef int* intpoint;  
intpoint p = new int;
```

- **p** được sử dụng như một con trỏ kiểu int

A 3D rendered yellow figure, resembling a stylized person, is holding a large, rectangular, light-yellow sign. The sign has a thin gold border and contains the text 'Mảng & con trỏ' in a black, sans-serif font. The figure is positioned behind the sign, with its hands visible holding the top and bottom edges. The background is a plain, light yellow gradient. On the left side, there are several thin, dark, curved lines that appear to be decorative or represent motion. The overall style is clean and modern.

Mảng & con trỏ

MẢNG

➤ **Mảng:**

- Mảng một chiều là tập hợp các phần tử **có cùng kiểu dữ liệu** được lưu trữ liên tiếp nhau trên bộ nhớ ảo,
- Nếu mảng một chiều có một hoặc nhiều hơn một phần tử, địa chỉ của phần tử đầu tiên cũng chính là địa chỉ của mảng một chiều.
- Vì vậy mảng là một kiểu biến con trỏ
- Ví dụ:
 - `int a[10]`
 - `int *p;`
 - Cả a và p đều là biến con trỏ

MẢNG

➤ Truy xuất địa chỉ của mảng trong bộ nhớ

➤ Cú pháp:

&<tên mảng>

➤ Truy xuất địa chỉ của 1 phần tử của mảng

➤ Cú pháp:

&<tên mảng>[chỉ số] hoặc **<tên mảng> + <chỉ số>**

➤ Truy xuất giá trị của phần tử của mảng

➤ Cú pháp:

***(<tên mảng>+<chỉ số>)**

MẢNG

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6      int arr[] = { 32, 13, 66, 11, 22 };
7      //show address of arr in virtual memory
8      cout << &arr << endl;
9
10     //show address of the first element of arr
11     cout << &arr[4] << endl;
12     cout << arr+4 << endl;
13     cout << *(arr+4) << endl;
14
15     cout << "===== " << endl;
16     cout << arr << endl;
17     return 0;
18 }
```

0x6ffe00

0x6ffe10

0x6ffe10

22

=====

0x6ffe00

PHÉP GÁN CON TRỎ CHO MẢNG

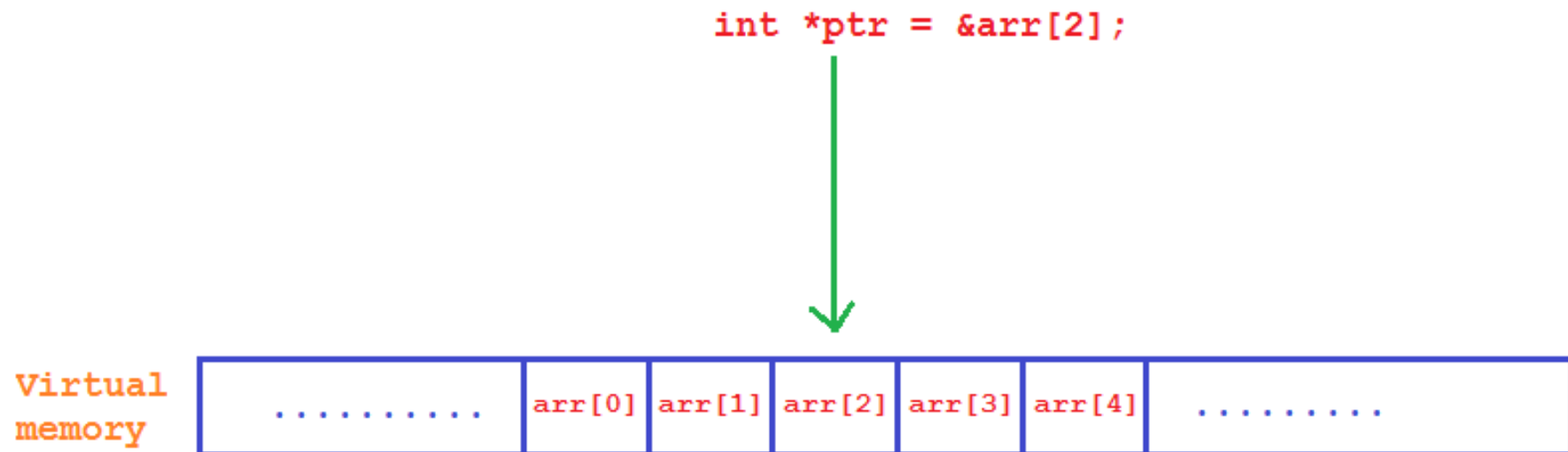
- Cú pháp: **<con trỏ> = <tên mảng>**
- Ví dụ:
 - `int a[10]`
 - `int *p;`
 - Cả a và p đều là biến con trỏ
- Phép gán: `p=a` → hợp lệ //p trỏ tới nơi a trỏ
- Phép gán: `a=p` → không hợp lệ //con trỏ mảng là một con trỏ hằng (mảng đã được cấp phát bộ nhớ; biến mảng luôn phải trỏ tới đó, không thể thay đổi)

PHÉP GÁN CON TRỎ CHO MẢNG

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {  int a[100], n, *p;
5      cout<<"Cho n = "; cin>>n;
6      cout<<"Nhap mang so:"<<endl;
7      for(int i = 0; i < n; i++){
8          cout<<"a["<<i<<"] = ";cin>>a[i];
9      }
10     cout<<"\n1. In mang:";
11     for(int i = 0; i < n; i++)
12         cout<<" "<<a[i];
13     //dung con tro p
14     cout<<"\n2. In theo con tro:";
15     p = a;
16     for(int i=0; i< n; i++)
17         cout<<" "<<p[i];
18     cout<<"\n3. In theo con tro:";
19     for(p = a; p< a+n; p++)
20         cout<<" "<<*p;
21     return 0;
22 }
```


CON TRỎ TRỎ VÀO PHẦN TỬ CỦA MẢNG

➡ Cú pháp: ***<con trỏ> = &<tên mảng>[chỉ số]**



CON TRỎ TRỞ VÀO PHẦN TỬ CỦA MẢNG

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6      int arr[] = { 32, 13, 66, 11, 22 };
7
8      int *ptr = &arr[2]; //ptr point to the 3rd element
9
10     cout << *ptr << endl;
11     cout << *(ptr - 1) << endl; //access the second element of arr
12     cout << *(ptr + 2) << endl; //access the last element of arr
13
14     return 0;
15 }
```

66
13
22

MẢNG ĐỘNG

➤ Hạn chế của mảng:

- Phải khai báo (xác định) kích thước
- Phải ước lượng kích thước lớn nhất cần thiết, thường lãng phí bộ nhớ hoặc không thể ước lượng

➤ Mảng động:

- Là mảng có kích thước không được chỉ ra tại thời điểm lập trình
- Kích thước của mảng được quyết định khi chương trình chạy
- Kích thước có thể được tăng thêm hoặc co lại khi cần -> tránh lãng phí bộ nhớ

MẢNG ĐỘNG

➡ Cú pháp:

<kiểu dữ liệu> *<con trỏ> = new <kiểu dữ liệu>[chỉ số];

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int *p_arr = new int[10];
7
8      //using this memory area
9      for (int i = 0; i < 10; i++)
10     {
11         //Set value for each element
12         cout<<"a["<<i<<"]=";
13         cin >> *(p_arr+i);
14     }
15
16     return 0;
17 }
```

```
a[0]=5
a[1]=4
a[2]=5
a[3]=6
a[4]=7
a[5]=8
a[6]=9
a[7]=5
a[8]=4
a[9]=3
```

MẢNG ĐỘNG

➡ Tạo mảng động:

- ➡ Sử dụng toán tử **new**
- ➡ Cấp phát bằng biến con trỏ
- ➡ Xử lý giống như mảng tĩnh
- ➡ VD:

Double *p = new double[10]; // tạo mảng động kiểu double có 10 phần tử

MẢNG ĐỘNG

➡ Hủy mảng động:

- ➡ Do mảng động được tạo ra khi chạy chương trình => phải hủy mảng động sau khi dùng xong.
- ➡ Sử dụng toán tử **delete**
- ➡ Chú ý cặp dấu **[]** sau từ khóa delete, chỉ ra việc xóa toàn bộ những ô nhớ đã được cấp phát động, nếu không có thì chỉ xóa ô nhớ đầu tiên.

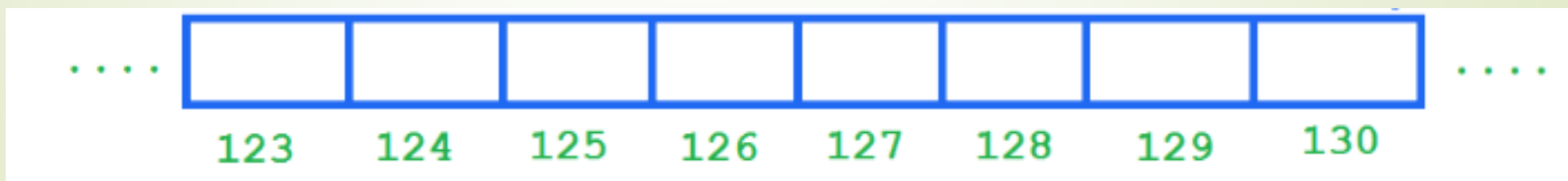
```
int *p = new int[10];  
//xu ly  
delete[] p;  
p = NULL;
```

TRUY CẬP ĐẾN CÁC PHẦN TỬ CỦA MẢNG ĐỘNG

Cách 1: Dùng chỉ số

```
int *p = new int [10];  
for (int i = 0; i < 10; i++)  
    p[i] = x;
```

- ➡ Truy cập tới ô nhớ tại địa chỉ (**p+i**) bằng cách viết **p[i]**



TRUY CẬP ĐẾN CÁC PHẦN TỬ CỦA MẢNG ĐỘNG

Cách 2: Không dùng chỉ số

```
int *p = new int [10];  
for (int i = 0; i < 10; i++)  
    *(p+i) = x;
```

- ➡ Dùng toán tử `*` để truy cập đến ô nhớ tại địa chỉ **(p+i)**

BÀI TẬP

Bài 1: Viết chương trình C++ làm những việc sau:

1. Nhập 10 giá trị nguyên
2. Tìm giá trị lớn nhất, nhỏ nhất.
3. Tìm giá trị có tần suất xuất hiện nhiều nhất.
4. Sắp xếp mảng theo thứ tự tăng dần và hiển thị kết quả.

A 3D rendered yellow figure stands behind a large, light-yellow rectangular sign. The figure's hands are visible, resting on the top and sides of the sign. The sign contains the text 'Truyền tham số là con trỏ vào hàm' in a black, sans-serif font. The background is white with some faint, thin grey lines on the left side.

Truyền tham số là
con trỏ vào hàm

TRUYỀN THAM SỐ LÀ CON TRỞ

- Ngôn ngữ lập trình C++ cho phép truyền một con trỏ tới một hàm (truyền địa chỉ cho hàm)
- Để truyền con trỏ tới hàm trong C++ ta chỉ cần khai báo tham số hàm có kiểu con trỏ.

TRUYỀN THAM SỐ LÀ CON TRỞ

➤ Truyền địa chỉ của biến vào hàm.

- Ví dụ: đổi giá trị của 2 biến được trỏ bởi con trỏ p1 và p2 cho nhau

```
1 void doigiatr(int *p1, int *p2)
2 {
3     int tg = *p1;
4     *p1 = *p2;
5     *p2 = tg;
6 }
```

TRUYỀN THAM SỐ LÀ CON TRỞ

- ➡ Truyền địa chỉ của mảng 1 chiều vào cho tham số kiểu con trỏ của hàm

```
1  #include <iostream>
2  using namespace std;
3  void nhap (int *q)
4  {   cout << "\nNhap day so:\n";
5      for (int i=0; i<10; i++)
6      {   cout<<"["<<i<<"] = ";
7          cin>>q[i];
8      }
9  }
10 void xuat (int *q)
11 {   for (int i=0; i<10; i++)
12     cout<<" "<<q[i];
13 }
```

VÍ DỤ

57

```
1  #include <iostream>
2  using namespace std;
3  void nhap (int *q)
4  {   cout << "\nNhap day so:\n";
5      for (int i=0; i<10; i++)
6      {   cout<<"["<<i<<"] = ";
7          cin>> *(q+i);
8      }
9  }
10 void xuat (int *q)
11 {   for (int i=0; i<10; i++)
12     cout<<" "<< *(q+i);
13 }
14 void Maxmin(int *q)
15 {   int max = q[0], min = q[0];
16     for (int i=1; i<10; i++)
17     {   if (q[i]>max) max = q[i];
18         if (q[i]<= min) min = q[i];
19     }
20     cout << "\nMAX = "<< max;
21     cout << "\nmin = "<< min;
22 }
```

```
23 void sapxep(int *q)
24 {   for (int i=0; i<9; i++)
25     for (int j=i+1; j<10; j++)
26         if (q[i]> q[j])
27         {int tg = q[i];
28             q[i] = q[j];
29             q[j] = tg;
30         }
31 }
32 int main()
33 {
34     int *p = new int[10] ;
35     nhap(p);
36     cout << "\nDay da nhap la:";
37     xuat(p);
38     Maxmin(p);
39     sapxep(p);
40     cout << "\nDay da sap xep la:";
41     xuat(p);
42     delete [] p;
43     p = NULL;
44 }
```


CON TRỎ HÀM

➡ Con trỏ có thể là kiểu trả về của hàm

➡ Cú pháp:

<kiểu trả về > *<ten hàm> (danh sách tham số>)

➡ Mảng **không** là kiểu trả về trong hàm được

➡ `Int [] hàm()` // Không hợp lệ

➡ Con trỏ có thể là kiểu trả về trong hàm.

Kết quả trả về là con trỏ.

➡ `Int * hàm ()` // Hợp lệ

```
23  int *sapxep(int *q)
24  {   for (int i=0; i<9; i++)
25      for (int j=i+1; j<10; j++)
26          if (q[i]> q[j])
27              {int tg = q[i];
28                  q[i] = q[j];
29                  q[j] = tg;
30              }
31      return q;
32  }
```

CON TRỎ HÀM

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int *nhap(int *a, int &n)
6  {
7      cout<<"n=";cin>>n;
8      a=new int[100];
9      for(int i=0;i<n;i++)
10     {
11         cout<<"a["<<i<<"]=";
12         cin>>*(a+i);
13     }
14     return a;
15 }
16
17 int main() {
18     int n;
19     int *x;
20
21     //Nhập mảng
22
23     x=nhap(x, n);
24     return 0;
25 }
```

```
n=5
a[0]=2
a[1]=3
a[2]=5
a[3]=6
a[4]=7
```

BÀI TẬP

Bài 2: Sử dụng con trỏ để:

1. Nhập vào 1 danh sách tên các sinh viên.
2. Sắp xếp danh sách đó theo thứ tự tăng dần.
3. In ra danh sách sau khi sắp xếp

BÀI TẬP

Bài 3: Viết các hàm để thực hiện các phép toán trên mảng một chiều bằng cách dùng **mảng động**:


- 1. Nhập số phần tử của mảng và các giá trị của phần tử trong mảng
- 2. Hiển thị các phần tử của mảng
- 3. Tính tổng các phần tử của mảng
- 4. Thêm phần tử x vào sau phần tử ở vị trí k của mảng với x, k nhập từ bàn phím.
- 5. Xóa khỏi mảng có phần tử có giá trị lớn nhất.

BÀI TẬP

- ➡ Sử dụng con trỏ và hàm để làm các bài tập sau:

Bài 4 : Viết chương trình nhập dãy A có n số thực. Nhập vào một số thực x , Tách dãy A thành 2 dãy con, dãy thứ nhất gồm toàn những phần tử nhỏ hơn x , dãy thứ hai gồm những phần tử còn lại. In hai dãy con ra màn hình.

Bài 5 : Đa thức bậc n : $P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ có thể được lưu trữ trong một vector (mảng) các hệ số $(a_0, a_1, a_2, \dots, a_n)$. Hãy viết chương trình nhập các hệ số một đa thức, và tính giá trị của đa thức bậc n trên theo giá trị x (cũng được nhập từ bàn phím).

A 3D rendered yellow figure stands behind a large, light-yellow rectangular sign. The figure's hands are visible, holding the sign from behind. The sign contains the text 'Một số thuật toán sắp xếp' in a black, sans-serif font. The background is white with some faint, thin grey lines on the left side.

Một số thuật
toán sắp xếp

MỘT SỐ THUẬT TOÁN SẮP XẾP

- Phương pháp Đổi chỗ trực tiếp (Interchange sort)
- Phương pháp Nổi bọt (Bubble sort)
- Phương pháp Chèn trực tiếp (Insertion sort)
- Phương pháp Chọn trực tiếp (Selection sort)

ĐỔI CHỖ TRỰC TIẾP (INTERCHANGE SORT)

So sánh các phần tử trong mảng. Nếu có 1 nghịch thế thì đổi chỗ

```
void Swap(int &a, int &b){  
    int temp = a;  
    a = b;  
    b = temp;  
}  
  
void InterchangeSort(int a[], int n){  
    for (int i = 0; i < n - 1; i++)  
        for (int j = i + 1; j < n; j++)  
            if(a[i] > a[j]) //nếu có nghịch thế thì đổi chỗ  
                Swap(a[i], a[j]);  
}
```

NỔI BỌT (BUBBLE SORT)

➤ Ý tưởng:

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đầu tiên của dãy hiện thời. Sau đó sẽ không xét nó ở bước tiếp theo
- Ở lần xử lý thứ l có vị trí đầu dãy là l
- Lặp lại cho đến khi mảng không còn cặp nào để xét

```
void BubbleSort(int a[], int n){  
    for (int i = 0; i < n - 1; i++)  
        for (int j = n - 1; j > i; j--)  
            if(a[j] < a[j-1])  
                Swap(a[j], a[j-1]);  
}
```

CHÈN (INSERTION SORT)

➤ Ý tưởng

- Tìm cách chèn phần tử $a[i]$ vào vị trí thích hợp của đoạn đã được sắp xếp để được dãy sắp xếp

```
void InsertionSort(int a[], int n){
    int pos, x;
    for(int i = 1; i < n; i++){ //đoạn a[0] đã sắp
        x = a[i];
        pos = i;
        while(pos > 0 && x < a[pos-1]){
            a[pos] = a[pos-1]; // dời chỗ
            pos--;
        }
        a[pos] = x;
    }
}
```

LỰA CHỌN (SELECTION SORT)

➤ Ý tưởng:

- Chọn phần tử nhỏ nhất trong n phần tử, đưa về vị trí đầu tiên.
- Dãy chưa sắp xếp gồm $n-1$ phần tử. Chọn phần tử nhỏ nhất trong đó và đưa về vị trí đầu tiên của dãy chưa sắp.

```
void SelectionSort(int a[], int n)
{
    int min; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
    for (int i = 0; i < n - 1; i++){
        min = i;
        for(int j = i + 1; j < n; j++){
            if (a[j] < a[min])
                min = j; // ghi nhận vị trí phần tử nhỏ nhất
        }
        if (min != i)
            Swap(a[min], a[i]);
    }
}
```