

# MỤC TIÊU

1

**Biết ý nghĩa và cách sử dụng  
con trỏ**

2

**Vận dụng được con trỏ vào hàm,  
mảng và cách cấp phát động**

- **Giới thiệu**
- **Sử dụng con trỏ**
- **Khai báo, khởi tạo con trỏ**
- **Con trỏ và địa chỉ**
- **Một số phép toán trên con trỏ**
- **Toán tử new và delete**
- **Con trỏ và mảng**
- **Hàm số có tham số con trỏ**



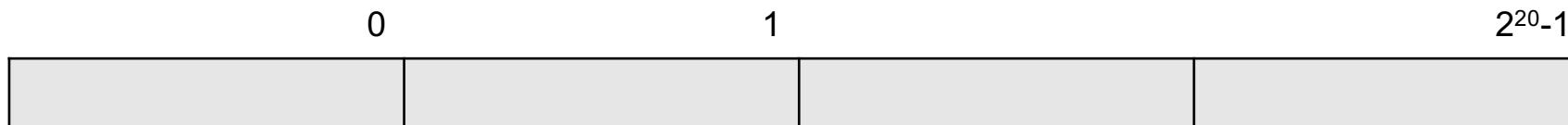
# CHƯƠNG 3: CON TRỎ

# GIỚI THIỆU

**Biến:** là tên được định danh để thay thế cho vị trí trong bộ nhớ.

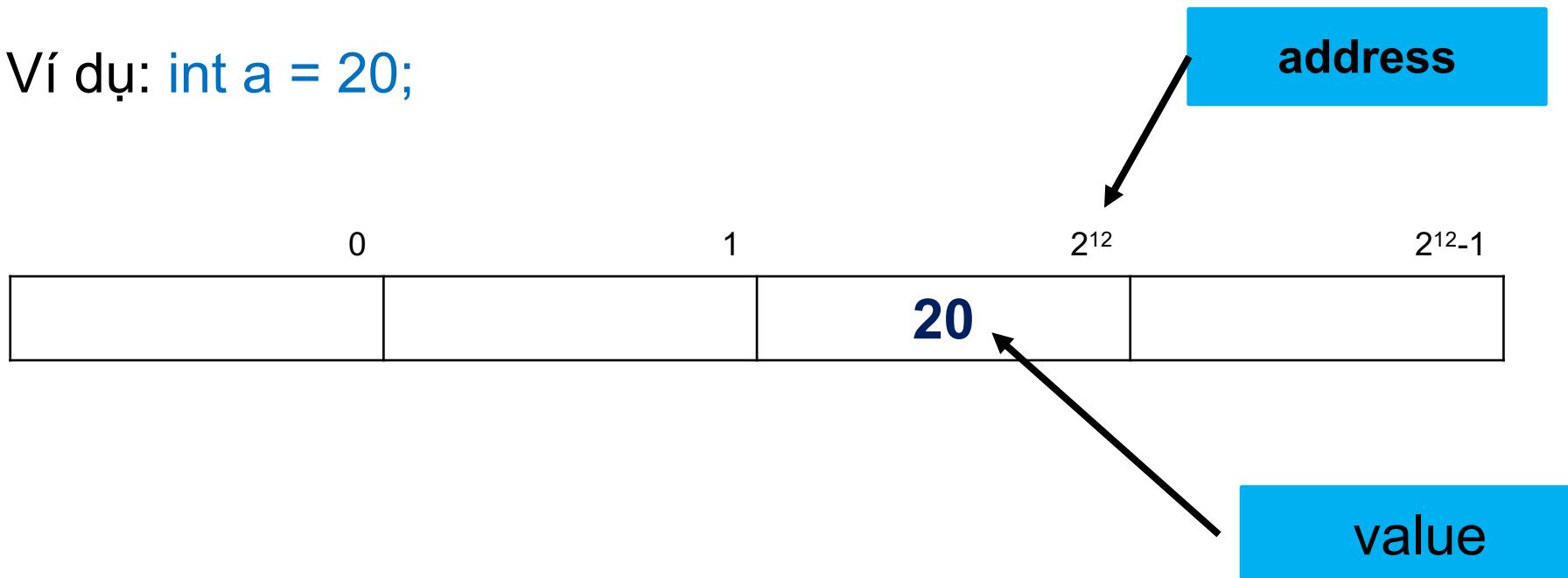
Bộ nhớ máy tính là chuỗi các bytes (bắt đầu từ 0). Các con số này chính là địa chỉ (address).

**Ví dụ: 1 MB (megabyte)**



# GIỚI THIỆU

Ví dụ: `int a = 20;`



Đối tượng lưu trữ địa chỉ (address) của biến là con trỏ (pointer)

- **Giới thiệu**
- **Sử dụng con trỏ**
- Khai báo, khởi tạo con trỏ
- Con trỏ và địa chỉ
- Một số phép toán trên con trỏ
- Toán tử new và delete
- Con trỏ và mảng
- Hàm số có tham số con trỏ



# CHƯƠNG 3: CON TRỎ

# SỬ DỤNG CON TRỎ

- Con trỏ rất phù hợp cho các vấn đề cần tiết kiệm bộ nhớ.
- Con trỏ có thể được sử dụng để truy cập vị trí của các phần tử trong mảng.
- Thích hợp với một số kiểu cấu trúc dữ liệu nâng cao.

# SỬ DỤNG CON TRỎ

- Một số ứng dụng cho thấy tính vượt trội của con trỏ so với các phương pháp thông thường

```
#include <iostream>

using namespace std;

int main() {
    int arr[100] = {1};
    // giải phóng bộ nhớ cho mảng ??
}
```

```
#include <iostream>

using namespace std;

int main() {
    cout << "Nhập vào n: ";
    int n;
    cin >> n;

    int *array = new int[n]; // cấp phát mảng động dùng con trỏ
    // sử dụng mảng
    // ...
    delete[] array; // giải phóng vùng nhớ mảng cho hệ điều hành
}
```

# SỬ DỤNG CON TRỎ

- Một số ứng dụng cho thấy tính vượt trội của con trỏ so với các phương pháp thông thường

```
// struct là kiểu cấu trúc dữ liệu giúp định nghĩa
// một tập hợp gồm nhiều kiểu dữ liệu cơ bản
struct node {
    int data;
    struct node *next; // pointer trỏ tới node chứa giá trị tiếp theo
};
```

# SỬ DỤNG CON TRỎ

- Một số ứng dụng cho thấy tính vượt trội của con trỏ so với các phương pháp thông thường

```
int arr[500] // có khả năng không dùng hết 500 phần tử  
  
int n;  
cin >> n; // n có thể được nhập từ bàn phím  
int arr* = new int[n]  
// dùng con trỏ tự điều chỉnh số phần tử phù hợp với bài toán
```

- Giới thiệu
- Sử dụng con trỏ
- **Khai báo, khởi tạo con trỏ**
- Con trỏ và địa chỉ
- Một số phép toán trên con trỏ
- Toán tử new và delete
- Con trỏ và mảng
- Hàm số có tham số con trỏ



# CHƯƠNG 3: CON TRỎ

# KHAI BÁO, KHỞI TẠO CON TRỎ

?] Cú pháp khai báo biến con trỏ:

**dataType\* variableName;**

?] Trong đó:

- dataType: kiểu dữ liệu của biến mà con trỏ trỏ đến
- variableName: tên biến con trỏ.

?] Ví dụ: **int\* p;**

- p: biến con trỏ tên p
- int: con trỏ p trỏ đến và lưu địa chỉ của dữ liệu kiểu số nguyên.

# KHAI BÁO, KHỞI TẠO CON TRỎ

?] Khởi tạo cho biến con trỏ:

- Biến con trỏ phải được gán giá trị trước khi sử dụng;
- Giá trị gán có thể chính là địa chỉ của 1 biến cụ thể hoặc là hằng NULL (0 hoặc nullptr);
- Con trỏ được gán NULL tức là con trỏ rỗng (null pointer).

?] **Lưu ý: Con trỏ không thể gán giá trị biến số thông thường.**

?] Ví dụ: 3 khai báo và khởi tạo sau đây là tương đương nhau

`int* ptr = NULL;`

`int* ptr = 0;`

`int* ptr = nullptr;`

# Khai báo, khởi tạo con trỏ

Một số lưu ý khi khai báo con trỏ:

int\* p1,p2;

P2 là kiểu số nguyên

P1 là con trỏ lưu địa chỉ biến  
kiểu số nguyên

- Nên khai báo mỗi biến con trỏ trên 1 dòng.
- Địa chỉ của con trỏ không thể bị thay đổi
- Nên đặt tên con trỏ bắt đầu bằng từ p hoặc ptr (pointer) để tiện kiểm soát.

# KHAI BÁO, KHỞI TẠO CON TRỎ

?] Toán tử & (address-of operator):

lấy địa chỉ của 1 biến

Ví dụ :

```
int x = 20;
int* p;
p = &x;
cout << "Dia chi cua bien x la: " << p << endl;
```

Địa chỉ của biến x là: 0x7ffe4289e6ec

Ví dụ 2:

```
double x = 20;
int* p;
p = &x; //error
cout << "Dia chi cua bien x la: " << p << endl;
```

# KHAI BÁO, KHỎI TẠO CON TRỎ

- ② Toán tử \* (content-of operator/ dereferences operator): lấy nội dung của biến trả đến.
- ② Ví dụ :

```
Gia tri cua bien x la: 20
```

```
int x = 20;
int* p;
p = &x;
cout << "Gia tri cua bien x la: " << *p << endl;
```

# KHAI BÁO, KHỞI TẠO CON TRỎ

?] Ví dụ :

```
int x = 20;  
int* p;  
p = &x;  
*p = 50;  
cout << "Gia tri cua bien x la: " << x << endl;
```

Gia tri cua bien x la: 50



- Giới thiệu
- Sử dụng con trỏ
- Khai báo, khởi tạo con trỏ
- **Con trỏ và địa chỉ**
- Một số phép toán trên con trỏ
- Toán tử new và delete
- Con trỏ và mảng
- Hàm số có tham số con trỏ



# CHƯƠNG 3: CON TRỎ

# CON TRỎ VÀ ĐỊA CHỈ

⇒ Đối với bộ nhớ, đơn vị nhỏ nhất lưu trữ là bytes

- Nếu ta có 8GB RAM thì ta có tổng cộng 8,589,934,592 ô nhớ lưu trữ.
- Mỗi ô nhớ sẽ này sẽ được đánh số vị trí.
- Các quá trình trích xuất, lưu trữ thông tin dữ liệu trên máy tính đều được thực hiện trên các ô nhớ này.
- Một số kiểu địa chỉ bộ nhớ điển hình: 0x008888ff, 0x001002fe...

# CON TRỎ VÀ ĐỊA CHỈ

- ② Con trỏ là một loại dữ liệu đặc biệt, giá trị lưu trong con trỏ luôn là địa chỉ của một biến số hay ô nhớ nào đó.
- ② Giá trị của con trỏ rất lớn
  - Các chương trình 32bit sẽ đánh địa chỉ từ 0 đến  $2^{32}-1$  với 4 bytes để lưu giá trị con trỏ
  - Các chương trình 64bit sẽ đánh địa chỉ từ 0 đến  $2^{64}-1$  với 8 bytes để lưu giá trị con trỏ

# CON TRỎ VÀ ĐỊA CHỈ

?] Một số lưu ý khi làm việc với con trỏ:

- Con trỏ có thể gán và trỏ trực tiếp vào địa chỉ do đó ta có thể trỏ tới bất cứ biến nào trong chương trình.
  - Lưu ý con trỏ không thể trỏ tới vùng nhớ của các chương trình khác vì mỗi một chương trình sẽ có một vùng nhớ ảo riêng.
- Khi tạo ra con trỏ mà không khởi tạo một địa chỉ thì con trỏ này sẽ được gán một giá trị vô định

# CON TRỎ VÀ ĐỊA CHỈ

```
int *p  
// con trỏ sẽ gán giá trị ngẫu nhiên  
// ví dụ địa chỉ là 0x040203ff
```

- ❑ Giả sử con trỏ p không được gán giá trị, nó sẽ được gán giá trị địa chỉ bất kỳ. Có 2 trường hợp xảy ra:
  - ❑ Nếu địa chỉ ngẫu nhiên đó chưa được cấp sử dụng thì ta sẽ có một ô nhớ ngẫu nhiên.
  - ❑ Nếu địa chỉ ngẫu nhiên đó đã được cấp sử dụng, con trỏ đang trích xuất với quyền ngang hàng (trừ quyền admin) thì **chương trình sẽ bị crash ngay lập tức.**

- Giới thiệu
- Sử dụng con trỏ
- Khai báo, khởi tạo con trỏ
- Con trỏ và địa chỉ
- **Một số phép toán trên con trỏ**
- Toán tử new và delete
- Con trỏ và mảng
- Hàm số có tham số con trỏ



# CHƯƠNG 3: CON TRỎ

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

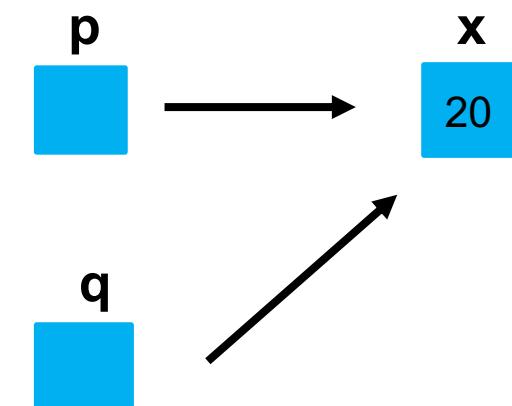
Phép gán:

- Gán địa chỉ 1 biến khác cho biến con trỏ (cùng kiểu dữ liệu)
- Gán 2 biến con trỏ cùng kiểu dữ liệu với nhau.

Ví dụ:

```
int x = 20;
int* p, *q;
p = &x;
q = p;
cout << "Gia tri luu tru trong q la: " << *q <<
endl;
```

Gia tri luu tru trong q la: 20



# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

Phép so sánh:

- 2 biến con trỏ cùng kiểu dữ liệu có thể so sánh được với nhau.

`int* ptr1;`

`Int* ptr2;`

Lệnh so sánh	Kết quả
<code>ptr1==ptr2</code>	TRUE nếu hai con trỏ cùng trỏ đến một vùng nhớ
<code>ptr1!=ptr2</code>	TRUE nếu hai con trỏ không cùng trỏ đến một vùng nhớ
<code>ptr1!=ptr2</code>	TRUE nếu con trỏ ptr1 khác NULL

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép so sánh

Ví dụ:

```
int* p, *q;  
p = NULL;  
int x = 2;  
q = &x;  
bool kq = p == q;  
cout << kq << endl;
```



0

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép so sánh

Ví dụ:

```
int* p, *q;  
p = NULL;  
int x = 2;  
q = &x;  
bool kq = p != q;  
cout << kq << endl;
```

1

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép so sánh

Ví dụ:

```
int* p, *q;  
int x = 2;  
p = &x;  
q = &x;  
bool kq = p == q;  
cout << kq << endl;
```

1

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép cộng, trừ

- Có thể cộng (+), trừ (-) giữa 2 biến con trỏ cùng kiểu dữ liệu, hay giữa con trỏ và 1 số nguyên.
- Tăng (++), giảm (--) tương tự như biến bình thường, nhưng giá trị sẽ thay đổi theo kích thước kiểu dữ liệu của con trỏ.

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép cộng, trừ

Ví dụ:

```
int *p;
int x = 2;
p = &x;
cout << "Dia chi " << p << " luu gia tri " << *p << endl;
// Dia chi con tro p tro toi la: 0x7ffc61ac3bbc
p++;
cout << "Dia chi " << p << " luu gia tri " << *p << endl;
// Dia chi con tro p tro toi sau khi tang la: 0x7ffc61ac3bc0
```

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

## Phép cộng, trừ

Ví dụ:

```
int *p;
int x = 2;
p = &x;
cout << "Dia chi: " << p << " luu gia tri: " << *p << endl;
// Dia chi: 0x7ffcdhb531ec luu gia tri: 2
p += 2;
cout << "Dia chi: " << p << " luu gia tri: " << *p << endl;
// Dia chi: 0x7ffcdhb531f4 luu gia tri: 32764
```

# MỘT SỐ PHÉP TOÁN VỚI CON TRỎ

**Lưu ý:** Không được sử dụng biến con trỏ trỏ đến hằng

Ví dụ:

```
const int x = 20;
int* p = &x; //error
• Không nên lạm dụng con trỏ, sẽ làm câu lệnh phức tạp thêm
int x = 20;
int c = *(&x);
cout << c;
```

- Giới thiệu
- Sử dụng con trỏ
- Khai báo, khởi tạo con trỏ
- Con trỏ và địa chỉ
- Một số phép toán trên con trỏ
- **Toán tử new và delete**
- Con trỏ và mảng
- Hàm số có tham số con trỏ



# CHƯƠNG 3: CON TRỎ

# TOÁN TỬ NEW VÀ DELETE

- Là biến được tạo trong khi chương trình đang thực thi.
- Cần biến cấp phát động ta dùng toán tử new.
- Không cần sử dụng nữa thì dùng toán tử delete (hủy bỏ biến đã tạo và thu hồi vùng nhớ).

# TOÁN TỬ NEW VÀ DELETE

**Toán tử new:** cấp phát cho biến đơn hoặc cho mảng

Cú pháp cấp phát biến đơn:

```
pointerName = new datatype;
```

Trong đó:

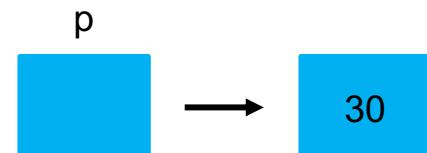
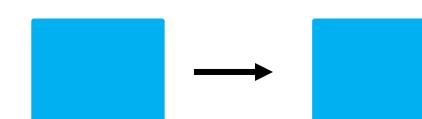
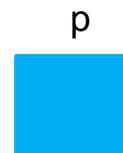
- pointername: tên con trỏ.
- dataType: kiểu dữ liệu mà biến pointerName trỏ đến.

Ví dụ:

```
int* p;
```

```
p = new int;
```

```
*p = 30;
```



# TOÁN TỬ NEW VÀ DELETE

Toán tử **new**: cấp phát cho biến đơn hoặc cho mảng

Cú pháp cấp phát mảng:

```
pointerName = new dataType[numberOfElement];
```

Trong đó:

- **pointerName**: tên con trỏ
- **dataType**: kiểu dữ liệu mà biến pointerName trỏ đến
- **NumberOfElement**: số lượng phần tử của mảng cấp phát động.

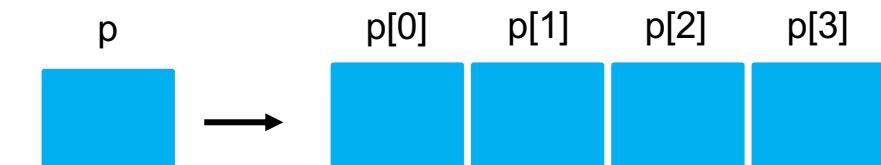
Ví dụ:

```
int* p;
```

```
int n;
```

```
cin>>n; //giả sử n = 4
```

```
p = new int[n];
```



# TOÁN TỬ NEW VÀ DELETE

- ② Toán tử delete: thu hồi vùng nhớ đã cấp phát khi không còn sử dụng (vùng nhớ của biến cấp phát động luôn phải được trả lại khi không còn sử dụng trong chương trình).

Hủy bỏ cấp phát động cho biến:

```
delete pointerName;
```

Hủy bỏ cấp phát động cho mảng:

```
delete [ ] pointerName;
```

# TOÁN TỬ NEW VÀ DELETE

Lưu ý khi cấp phát động:

`int* p;`

`p = new int;`

`*p = 50;`

`p = new int;`

`*p = 35;`

- Không thể truy xuất đến biến đầu tiên lưu trữ giá trị 50.
- Nên hủy biến cấp phát động p ban đầu: `delete p` rồi hãy tiếp tục cấp phát động cho biến mới.

# TOÁN TỬ NEW VÀ DELETE

Cấp phát động cho mảng 1 chiều

Cấp phát tĩnh	Cấp phát động
<ul style="list-style-type: none"><li>Khai báo mảng với số lượng phần tử tối đa</li><li>Nhập số lượng phần tử thực tế làm việc</li><li>Nhập giá trị cho số lượng phần tử thực tế</li><li>Thực hiện tính toán</li></ul>	<ul style="list-style-type: none"><li>Nhập số lượng phần tử cần</li><li>Xin cấp phát đúng số lượng phần tử cần cho mảng</li><li>Nhập giá trị cho đúng số phần tử của mảng trong vùng nhớ</li><li>Thực hiện tính toán</li><li>Hủy vùng nhớ cấp phát</li><li>Đưa con trỏ về con trỏ rỗng</li></ul>

# TOÁN TỬ NEW VÀ DELETE

Mảng 1 chiều cấp phát động

```
int n;
cout << "Nhập số lượng phần tử cần: ";
cin >> n;
int* a;
a = new int[n];
//Nhập mảng
for (int i = 0; i < n; i++)
{
    cout << "Nhập giá trị cho phần tử thứ " << i + 1
    << ": ";
    cin >> a[i];
}
```

# TOÁN TỬ NEW VÀ DELETE

Mảng 1 chiều cấp phát động

```
//Xuat mang
cout << "\nMang luu tru la: ";
for (int i = 0; i < n; i++)
    cout << a[i] << "\t";
cout << endl;
//Huy cap phat dong, dua con tro ve con tro rong
delete []a;
a = nullptr;
```

# TOÁN TỬ NEW VÀ DELETE

Cấp phát động cho mảng 2 chiều

Cấp phát tĩnh	Cấp phát động
<ul style="list-style-type: none"><li>Khai báo mảng với số phần tử tối đa cho mỗi chiều</li><li>Nhập số lượng phần tử thực tế làm việc ở mỗi chiều</li><li>Nhập giá trị cho số lượng phần tử thực tế tương ứng</li><li>Thực hiện tính toán</li></ul>	<ul style="list-style-type: none"><li>Nhập số lượng phần tử cần ở mỗi chiều</li><li>Xin cấp phát đúng số lượng phần tử cần cho mảng<ul style="list-style-type: none"><li>cấp phát 1 mảng các con trỏ(số dòng)</li><li>cấp phát vùng nhớ cho mỗi con trỏ trên từng dòng(số cột)</li></ul></li><li>Nhập giá trị cho từng phần của mảng trong vùng nhớ</li><li>Thực hiện tính toán</li><li>Hủy vùng nhớ cấp phát (theo vùng nhớ cho con trỏ rồi đến hủy vùng nhớ cho mảng)</li><li>Đưa con trỏ về con trỏ rỗng</li></ul>

# TOÁN TỬ NEW VÀ DELETE

Mảng 2 chiều cấp phát động

```
int r, c;
cout << "Nhập số dòng và số cột cần: "; cin >> r >> c;
int **a;
//Cấp phát mảng các con trỏ (số dòng)
a = new int* [r];
//Cấp phát vùng nhớ cho mỗi con trỏ (số cột)
for (int i = 0; i < r; i++)
a[i] = new int [c];
```

# TOÁN TỬ NEW VÀ DELETE

Mảng 2 chiều cấp phát động

```
//Nhập mảng
for (int i = 0; i < r; i++)
{
    cout << "\nNhập " << c << " giá trị cho dòng thứ " << i + 1 <<
    ": ";
    for (int j = 0; j < c; j++)
        cin >> a[i][j];
}
//Xuất mảng
cout << "\nMảng đang lưu trữ: " << endl;
for (int i = 0; i < r; i++)
{
    for (int j = 0; j < c; j++)
        cout << a[i][j] << "\t";
    cout << endl;
}
```

# TOÁN TỬ NEW VÀ DELETE

Mảng 2 chiều cấp phát động

---

```
//huy vung nho da cap phat
for (int i = 0; i < r; i++)
    delete [] a[i];
delete [] a;
a = nullptr;
```

---

# TOÁN TỬ NEW VÀ DELETE

Quy trình chuyển cấp phát động cho mảng 2 chiều bằng hàm

## ② Function prototype:

```
void nhap (int **a, int r, int c);
```

```
void xuat (int **a, int r, int c);
```

## ③ Hàm main:

- Nhập r, c.
- Cấp phát mảng con trỏ (số dòng).
- Cấp phát vùng nhớ cho mỗi con trỏ (số cột).
- Gọi hàm nhập.

- **Giới thiệu**
- **Sử dụng con trỏ**
- **Khai báo, khởi tạo con trỏ**
- **Con trỏ và địa chỉ**
- **Một số phép toán trên con trỏ**
- **Toán tử new và delete**
- **Con trỏ và mảng**
- **Hàm số có tham số con trỏ**



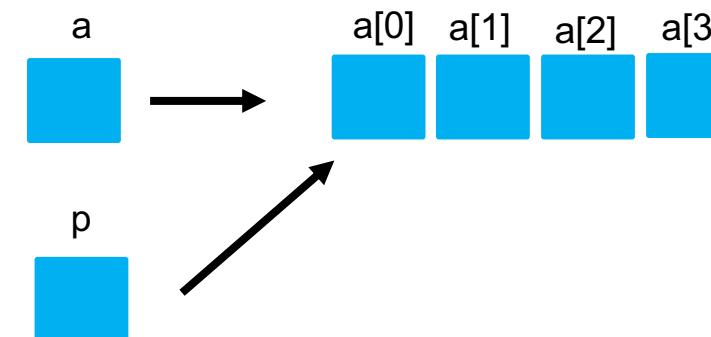
# CHƯƠNG 3: CON TRỎ

# CON TRỎ VÀ MẢNG

- Tên mảng là con trỏ trả đến phần tử đầu tiên trong mảng

Ví dụ:

```
int a[4];
int *p;
p = a;
Tương đương với
p = a[0];
```



# CON TRỎ VÀ MẢNG

Ví dụ cài đặt giữa mảng và con trỏ:

```
int a[] = {24, 31, 19, 16};  
int n = sizeof(a)/sizeof(a[0]);  
int *p = a;  
  
//in mang dung ten mang va ky phap chi so  
cout<<"Ten mang va ky phap chi so\n";  
for (int i=0;i<n;i++)  
    cout<<"a["<<i<<"] = "<<a[i]<<endl;  
  
//in mang dung ten mang va ky phap do doi  
cout<<"\nTen mang va ky phap do doi\n";  
for (int offset = 0; offset<n ;offset++)  
    cout<<*a( + "<<offset<<") = "  
    <<*(a + offset) <<endl;
```

# CON TRỎ VÀ MẢNG

Ví dụ cài đặt giữa mảng và con trỏ:

```
//in mang dung con tro va ky phap chi so
cout<<"\nCon tro va ky phap chi so\n";
for(int i=0;i<n;i++)
    cout<<p["<<i<<"] = "<<p[i]<<endl;
```

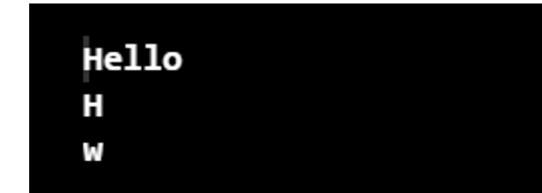
```
//in mang dung con tro va ky phap doi doi
cout<<"\nCon tro va ky phap doi doi\n";
for (int offset = 0; offset<n ;offset++)
    cout<<"*(p + "<<offset<<") = "
    <<*(p + offset) <<endl;
```

# CON TRỎ VÀ MẢNG

- Mảng con trỏ là một mảng lưu trữ toàn các biến con trỏ.
- Thường dùng cho mảng chứa đối tượng chuỗi, hay mảng chứa đối tượng là ký tự.

Ví dụ:

```
char *a[2];
a[0] = "Hello";
a[1] = "world";
cout << a[0] << endl;
cout << *a[0] << endl;
cout << *a[1] << endl;
```



- **Giới thiệu**
- **Sử dụng con trỏ**
- **Khai báo, khởi tạo con trỏ**
- **Con trỏ và địa chỉ**
- **Một số phép toán trên con trỏ**
- **Toán tử new và delete**
- **Con trỏ và mảng**
- **Hàm số có tham số con trỏ**



# CHƯƠNG 3: CON TRỎ

# HÀM SỐ CÓ THAM SỐ CON TRỎ

⇒ Được truyền theo 2 hình thức: tham trị và tham chiếu

- Tham trị: truyền địa chỉ.
- Tham chiếu: dùng dấu & sau dấu \* của con trỏ.

Thường được ưu tiên khi truyền có cấp phát động.

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- Tham số hàm có kiểu con trỏ **truyền theo tham trị**.

2, 1

```
void hoandoi(int* a, int* b)
{
    int tam = *a;
    *a = *b;
    *b = tam;
}
int main()
{
    int x = 1;
    int y = 2;
    int* p = &x;
    int* q = &y;
    hoandoi(p, q);
    cout << x << ", " << y << endl;
    return 0;
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

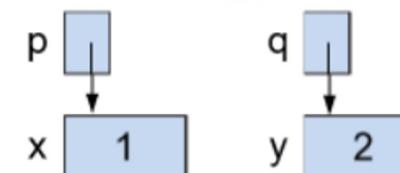
- Tham số hàm có kiểu con trỏ **truyền theo tham chiếu**.

2, 1

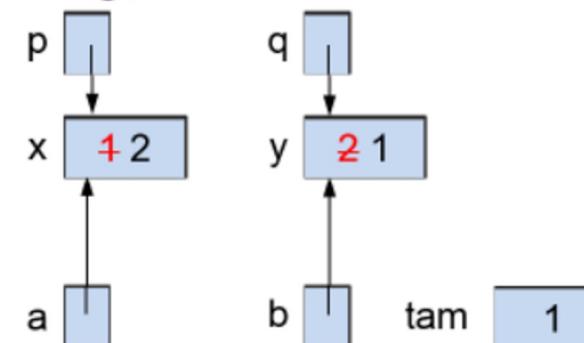
```
void hoandoi(int*& a, int*& b)
{
    int tam = *a;
    *a = *b;
    *b = tam;
}
int main()
{
    int x = 1;
    int y = 2;
    int* p = &x;
    int* q = &y;
    hoandoi(p, q);
    cout << x << ", " << y << endl;
    return 0;
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

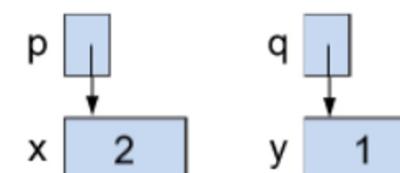
- Trước khi gọi hoandoi:



- Khi gọi hoandoi:



- Sau khi hoandoi thực hiện xong:



# HÀM SỐ CÓ THAM SỐ CON TRỎ

- Tham số hàm có kiểu con trỏ truyền **KHÔNG** theo tham chiếu.

```
void nhap(int* a, int n)
{
    a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }
}
void xuat(const int* a, int n)
{
    cout << "\nCac phan tu trong mang la: ";
    for (int i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl << endl;
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- ?] Gọi thực thi trong hàm main():

```
int main()
{
    int* p;
    int n;
    cout << "Nhập số phần tử: ";
    cin >> n;
    p = new int[n];
    nhap(p, n);
    xuat(p, n);
}
```

```
Nhập số phần tử: 5
a[0]= 3
a[1]= 5
a[2]= 1
a[3]= 3
a[4]= 2

Các phần tử trong mảng là: -842150451 -842150451 -842150451 -842150451 -842150451
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- ☐ Nếu cấp phát động không ở trong hàm nhập khi truyền mảng (con trỏ) theo kiểu tham trị thì mảng xuất bình thường (nhờ vào giá trị địa chỉ).

```
Nhap so phan tu: 2
a[0]= 6
a[1]= 7

Cac phan tu trong mang la: 6 7
```

```
void nhap(int* a, int n)
{
    //a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- Tham số hàm có kiểu con trỏ truyền theo tham chiếu

```
void nhap (int*& a, int n);
```

```
void xuat (int* a, int n);
```

```
void nhap(int*& a, int n)
{
    a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- ?] Gọi thực thi trong hàm main():

```
int main()
{
    int* p;
    int n;
    cout << "Nhập số phần tử: ";
    cin >> n;
    p = new int[n];
    nhap(p, n);
    xuat(p, n);
}
```

```
Nhập số phần tử: 3
a[0]= 1
a[1]= 2
a[2]= 3

Các phần tử trong mảng là: 1 2 3
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- ☒ Hàm trả về con trỏ

Hàm trả về giá trị nhỏ nhất của 1 mảng số nguyên bằng cách dùng con trỏ.

```
//ham tim phan tu nho nhat trong mang
//nhan vao: mang so nguyen a, so phan tu n
// tra ve: con tro den phan tu nho nhat
double* nhonhat(double a[], int n)
{
    int vitri_nhonhat=0;
    for(int i=1;i<n;i++)
        if(a[i]<a[vitri_nhonhat])
            vitri_nhonhat=i;
    return &a[vitri_nhonhat];
}
```

# HÀM SỐ CÓ THAM SỐ CON TRỎ

- ☒ Hàm trả về con trỏ

Hàm trả về giá trị nhỏ nhất của 1 mảng số nguyên bằng cách dùng con trỏ.

```
int main()
{
    double arr[] = {11, 6, 45, 16, 9, 100, 231, 55};
    int arrsize = sizeof(arr) / sizeof(arr[0]);
    double* p = nhonhat(arr, arrsize);
    cout << "phan tu nho nhat la " << *p << endl;
    return 0;
}
```



**THANK YOU**