

# CHƯƠNG 3. LỚP VÀ ĐỐI TƯỢNG

ThS. Trần Anh Dũng

C++



Microsoft®

Visual Studio®

# Nội dung

- ❖ Cú pháp khai báo lớp
- ❖ Định nghĩa hàm thành phần của lớp
- ❖ Khai báo và tạo lập đối tượng
- ❖ Phạm vi truy xuất
- ❖ Phương thức thiết lập – Constructor
- ❖ Phương thức hủy bỏ – Destructor
- ❖ Phương thức Truy vấn, Cập nhật
- ❖ Thành viên tĩnh – static member

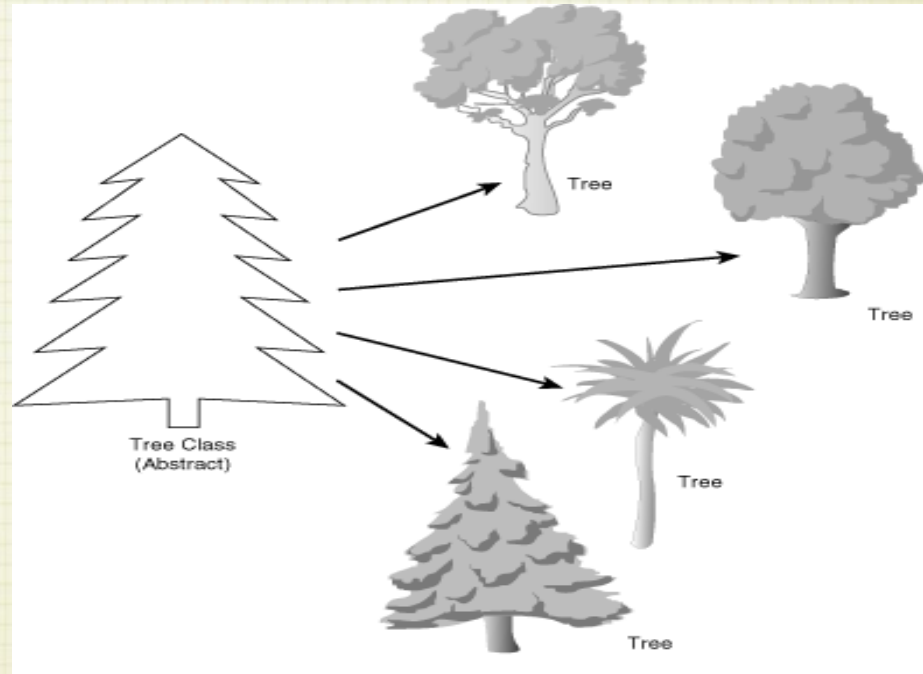
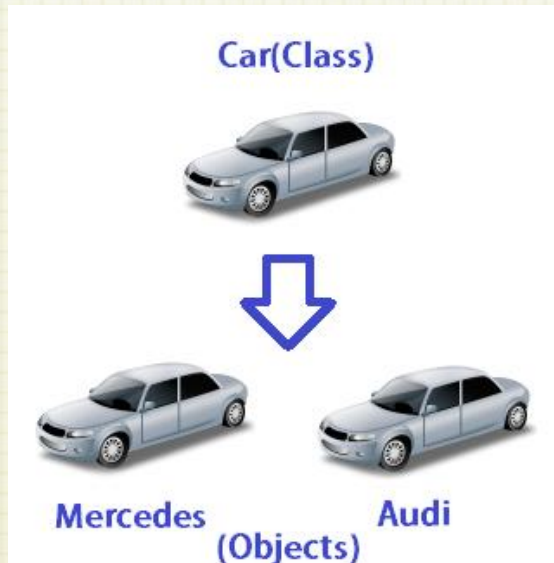
# Lớp trong C++

- ❖ Một lớp bao gồm các thành phần **dữ liệu (thuộc tính)** và các **phương thức (hàm thành phần)**.
- ❖ Lớp trong C++ thực chất là một kiểu dữ liệu do người sử dụng định nghĩa.
- ❖ Trong C++, dùng từ khóa **class** để chỉ điểm bắt đầu của một lớp sẽ được cài đặt.



# Lớp đối tượng

- ❖ Lớp là một mô tả trừu tượng của nhóm các đối tượng cùng bản chất, ngược lại mỗi một đối tượng là một **thể hiện** cụ thể cho những mô tả trừu tượng đó.



# Cú pháp khai báo lớp

```
class <tên_lớp>
```

```
{
```

```
//Thành phần dữ liệu
```

```
//Thành phần xử lý
```

```
};
```



```
class (class-name)
{
    private:
    variable declarations;

    public:
    function declarations;
};
```

# Cú pháp khai báo lớp

`class <tên_lớp> {`

`private:`

<khai báo thành phần riêng trong từng đối tượng>

`protected:`

<khai báo thành phần riêng trong từng đối tượng,  
có thể truy cập từ lớp dẫn xuất >

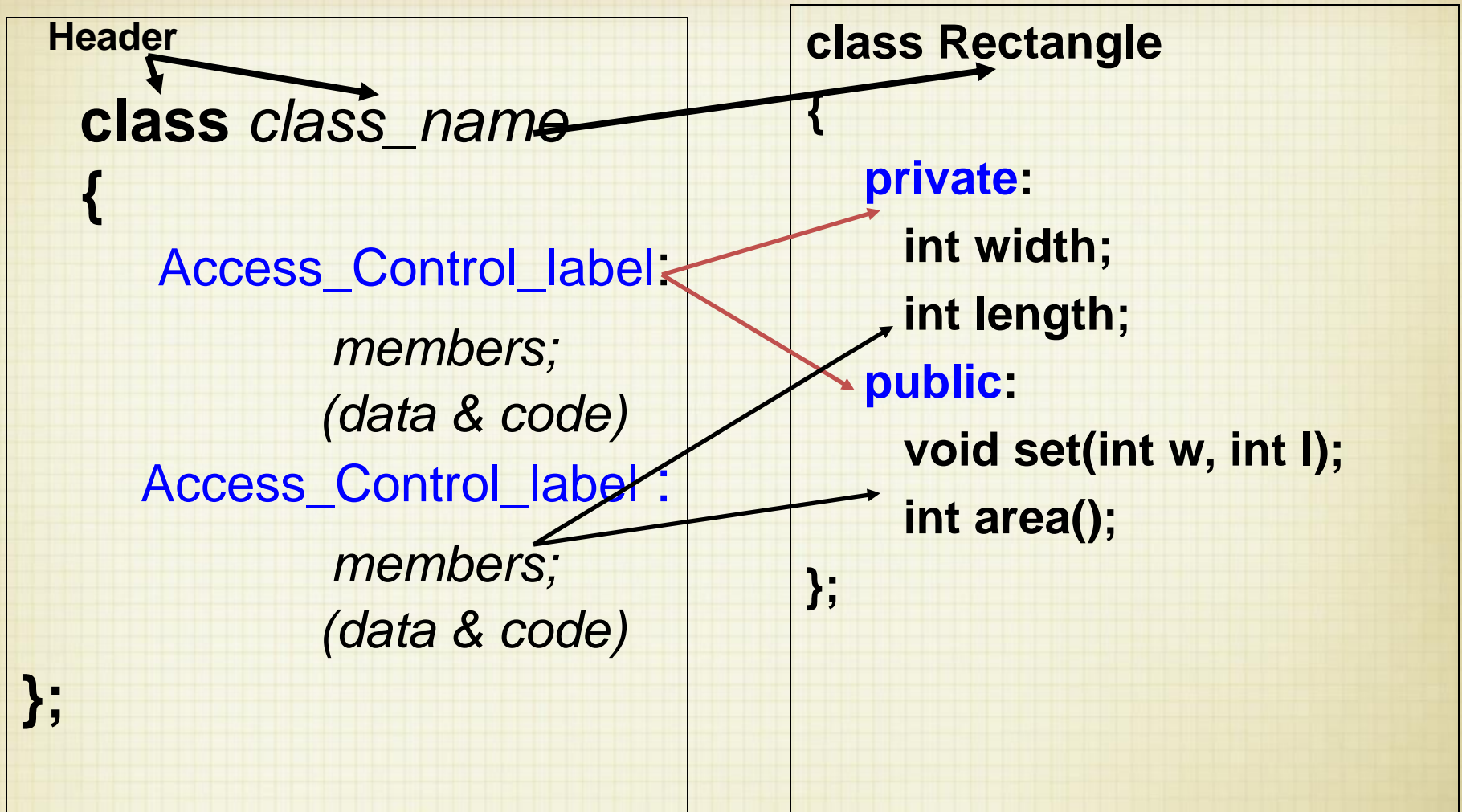
`public:`

<khai báo thành phần công cộng>

`};`



# Cú pháp khai báo lớp



# Các thành phần của lớp

- ❖ **Thuộc tính:** Các thuộc tính được khai báo giống như khai báo biến trong C
- ❖ **Phương thức:** Các phương thức được khai báo giống như khai báo hàm trong C. Có hai cách định nghĩa thi hành của một phương thức
  - ❖ Định nghĩa thi hành trong lớp
  - ❖ Định nghĩa thi hành ngoài lớp



# Cơ chế tạo lập các lớp

- ❖ Xác định các thuộc tính (dữ liệu)
  - Những gì mà ta biết về đối tượng – giống như một struct
- ❖ Xác định các phương thức (hành vi)
  - Những gì mà đối tượng có thể làm
- ❖ Xác định các quyền truy xuất

# Định nghĩa hàm thành phần

❖ Cú pháp định nghĩa các hàm thành phần ở bên ngoài khai báo lớp:

<tên kiểu giá trị trả về> <tên lớp>::<tên hàm> (<danh sách tham số>)

```
{  
    <nội dung >  
}
```

Ví dụ:

```
void point::display() {  
    //.....  
}
```

# Định nghĩa hàm thành phần

```
class Rectangle{  
    private:  
        int width, length;  
    public:  
        void set (int w, int l);  
        int area() { return width*length; }  
};
```

inline

class name

member function name

```
void Rectangle :: set (int w, int l)  
{  
    width = w;  
    length = l;  
}
```

scope operator

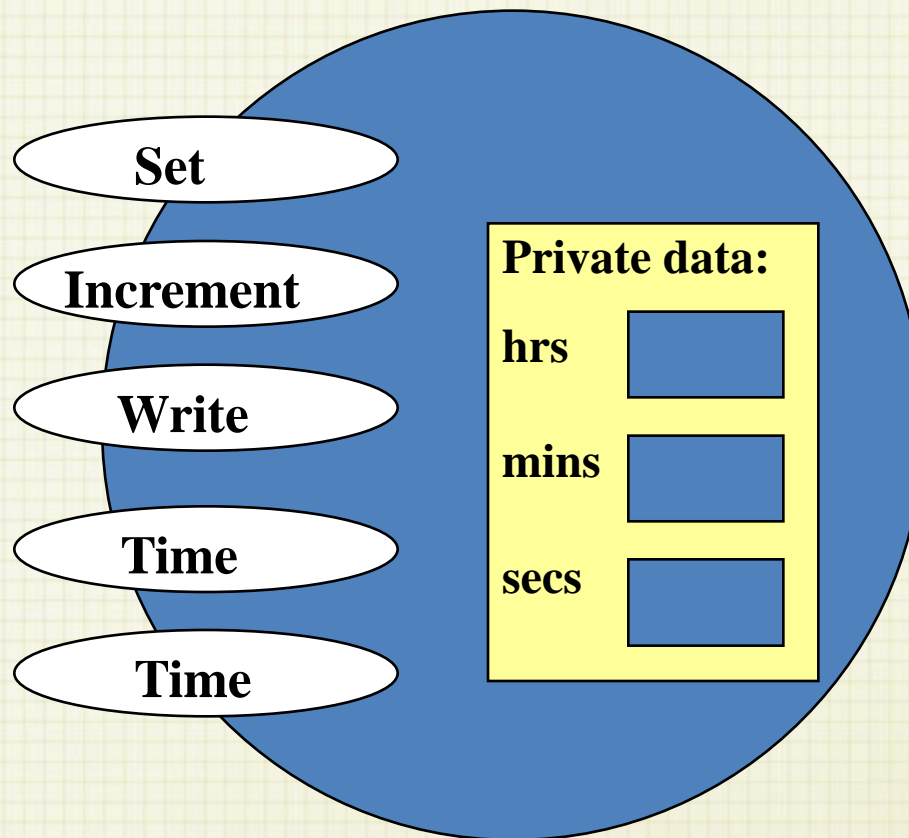


# Ví dụ lớp Time

```
class Time {  
    public:  
        void Set (int hours , int minutes , int seconds);  
        void Increment ( );  
        void Write ( ) const;  
        Time (int initHrs, int initMins, int initSecs ); //constructor  
        Time ( ); //default constructor  
    private:  
        int      hrs;  
        int      mins;  
        int      secs;  
};
```

# Ví dụ lớp Time

## Time class



# Khai báo và tạo lập đối tượng

❖ Khai báo và tạo đối tượng:

<tên lớp> <tên đối tượng>;

❖ Gọi hàm thành phần của lớp

<tên đối tượng>.<tên hàm thành phần> (<danh sách các tham số nếu có>);

<tên con trỏ đối tượng>→<tên hàm thành phần> (<danh sách các tham số nếu có>);



# Khai báo và tạo lập đối tượng

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

**r1** is statically allocated

```
void main()
{
    Rectangle r1;
    ➔ r1.set(5, 8);
}
```

**r1**

|                                       |
|---------------------------------------|
| <b>width = 5</b><br><b>length = 8</b> |
|---------------------------------------|

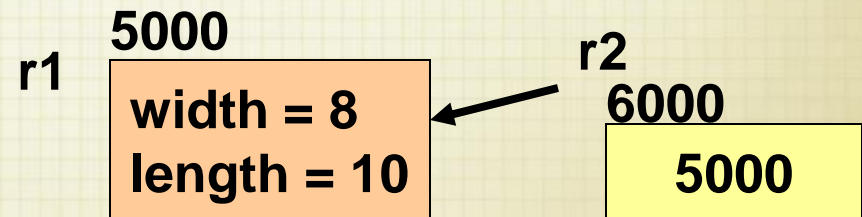
# Khai báo và tạo lập đối tượng

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

r2 is a pointer to a Rectangle object

```
main()
{
    Rectangle r1;
    r1.set(5, 8);           //dot notation

    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);         //arrow notation
}
```



# Khai báo và tạo lập đối tượng

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

r3 is dynamically allocated

```
main()
{
    Rectangle *r3;
    r3 = new Rectangle();

    r3->set(80,100); //arrow notation

    delete r3;
    ➡ r3 = NULL;
}
```

r3 6000  
NULL



# Ví dụ

❖ Xây dựng lớp **Điểm (Point)** trong hình học 2D

- Thuộc tính
  - Tung độ
  - Hoành độ
- Thao tác (phương thức)
  - Khởi tạo
  - Di chuyển
  - In ra màn hình
  - ...

# Ví dụ

```
/*point.cpp*/
#include <iostream>
using namespace std;
class point {
    /*khai báo các thành phần dữ liệu riêng*/
private:
    int x,y;
    /*khai báo các hàm thành phần công cộng*/
public:
    void init(int ox, int oy);
    void move(int dx, int dy);
    void display();
};
```

# Ví dụ

```
void point::init(int ox, int oy) {  
    cout<<"Ham thanh phan init\n";  
    x = ox; y = oy;  
    /*x,y là các thành phần của đối tượng gọi hàm thành phần*/  
}  
void point::move(int dx, int dy) {  
    cout<<"Ham thanh phan move\n";  
    x += dx; y += dy;  
}  
void point::display() {  
    cout<<"Ham thanh phan display\n";  
    cout<<"Toa do: "<<x<<" "<<y<<"\n";  
}
```



# Ví dụ

```
void main()
{
    point p;
    p.init(2,4); /*gọi hàm thành phần từ đối tượng*/
    p.display();
    p.move(1,2);
    p.display();
}
```

Hàm thành phần init

Hàm thành phần display

Toa do: 2 4

Hàm thành phần move

Hàm thành phần display

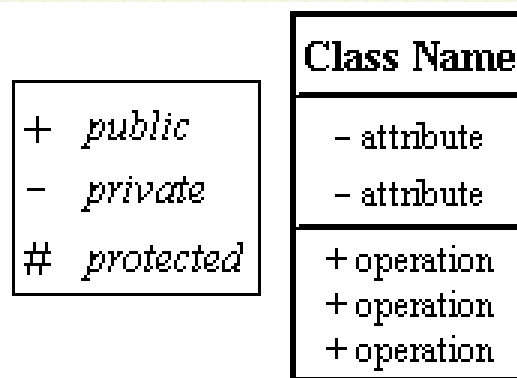
Toa do: 3 6

# Phạm vi truy xuất

- ❖ Trong định nghĩa của lớp ta có thể xác định **khả năng truy xuất thành phần** của một lớp nào đó từ bên ngoài phạm vi lớp.
- ❖ **private**, **protected** và **public** là các **từ khóa** xác định phạm vi truy xuất
- ❖ Mọi thành phần được liệt kê trong phần **public** đều có thể truy xuất trong **bất kỳ** hàm nào.
- ❖ Những thành phần được liệt kê trong phần **private** chỉ được truy xuất **bên trong phạm vi lớp**.

# Phạm vi truy xuất

- ❖ Trong lớp có thể có nhiều nhãn **private** và **public**
- ❖ Mỗi nhãn này có **phạm vi ảnh hưởng** cho đến khi gặp một nhãn kế tiếp hoặc hết khai báo lớp.
- ❖ Nhãn **private** đầu tiên có thể bỏ qua vì C++ ngầm hiểu rằng các thành phần trước nhãn **public** đầu tiên là **private**.





# Phạm vi truy xuất – Ví dụ

```
class TamGiac{  
    private:  
        float a,b,c;/*độ dài ba cạnh*/  
    public:  
        void Nhap();/*nhập vào độ dài ba cạnh*/  
        void In();/*in ra các thông tin liên quan đến tam giác*/  
    private:  
        int Loaitg();//cho biết kiểu của tam giác: 1-d,2-v,3-c,4-v,5-t  
        float DienTich();/*tính diện tích của tam giác*/  
};
```

# Phạm vi truy xuất – Ví dụ

```
class TamGiac{  
    private:  
        float a,b,c;/*độ dài ba cạnh*/  
        int Loaitg();//cho biết kiểu của tam giác: 1-d,2-v,3-c,4-v,5-t  
        float DienTich();/*tính diện tích của tam giác*/  
    public:  
        void Nhap();/*nhập vào độ dài ba cạnh*/  
        void In();/*in ra các thông tin liên quan đến tam giác*/  
};
```

# Tham số hàm thành phần

```
void point::init (int xx, int yy){  
    x = xx;  
    y = yy; //x, y là thành phần của lớp point  
}
```

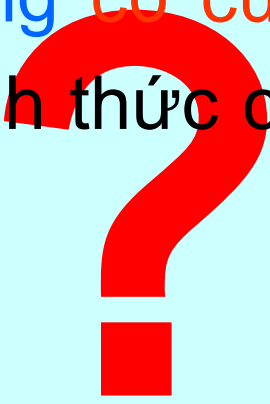


- ❖ **Hàm thành phần** có quyền truy cập đến các thành phần **private** của đối tượng gọi nó

# Tham số hàm thành phần

❖ Hàm thành phần (partial function) quyền truy cập đến tất cả các thành phần (private) của đối tượng, tham chiếu đối tượng hay con trỏ đối tượng có cùng kiểu lớp. Trung (pointing) là tham số hình thức của nó. return (x==pt→x && y==pt→y);

```
int point::Trung (point &pt) {  
    return (x==pt.x && y==pt.y);  
}
```





# Con trỏ this

- ❖ Từ khóa **this** trong định nghĩa của các hàm thành phần lớp dùng để xác định địa chỉ của đối tượng dùng làm **tham số ngầm định** cho hàm thành phần.
- ❖ Con trỏ **this** tham chiếu đến đối tượng đang gọi hàm thành phần.
- ❖ Ví dụ:

```
int Trung(point pt){  
    return (this → x == pt.x && this → y == pt.y);  
}
```

# Phép gán đối tượng

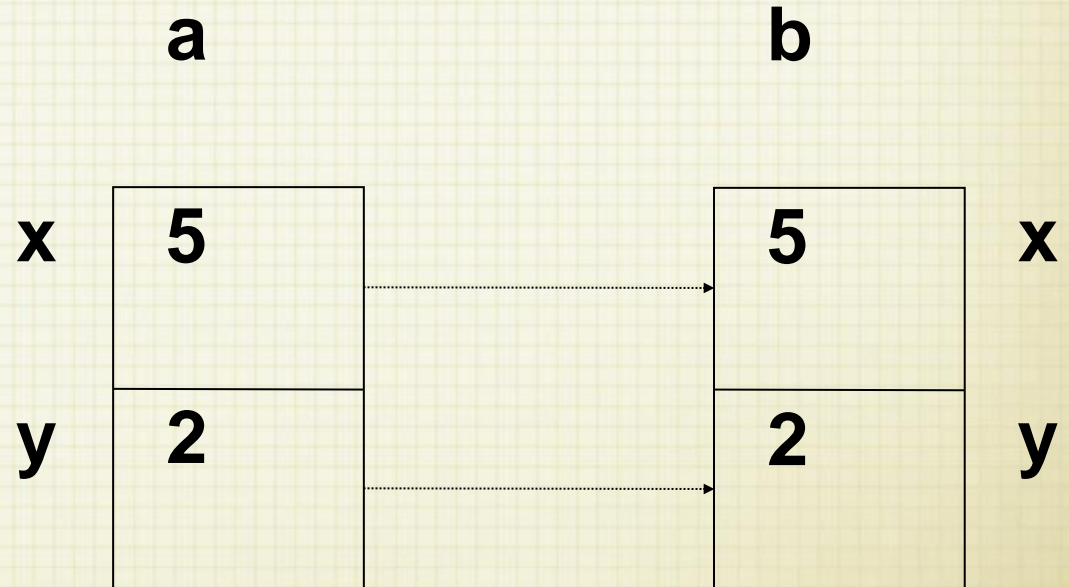
❖ Là việc sao chép giá trị các thành phần dữ liệu từ đối tượng **a** sang đối tượng **b** tương ứng từng đôi một

❖ Ví dụ:

```
point a, b;
```

```
a.init(5,2);
```

```
b = a;
```



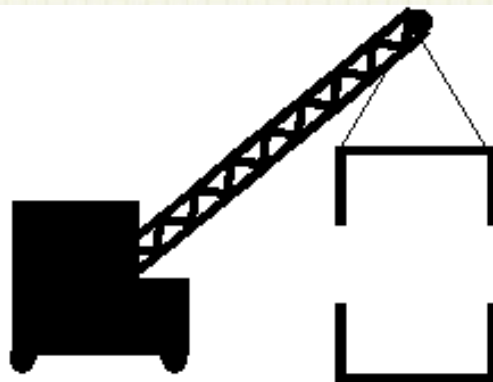
# Phương thức thiết lập

- ❖ Trong hầu hết các thuật giải, để giải quyết một vấn đề → thường phải thực hiện các công việc:
  - Khởi tạo giá trị cho biến, cấp phát vùng bộ nhớ của biến con trỏ, mở tập tin để truy cập,...
  - Hoặc khi kết thúc, chúng ta phải thực hiện quá trình ngược lại như: Thu hồi vùng bộ nhớ đã cấp phát, đóng tập tin,...



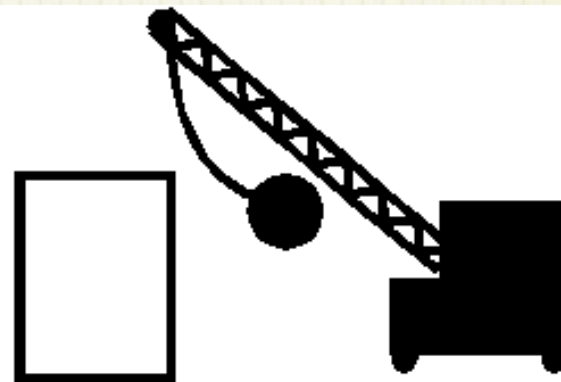
# Phương thức thiết lập

- ❖ Các ngôn ngữ OOP có các phương thức để thực hiện công việc này một cách “tự động” gọi là *phương thức thiết lập* và *phương thức hủy bỏ*.



Constructor

```
MyClass *MyObjPtr = new MyClass();
```



Destructor

```
delete MyObjPtr;
```



# Phương thức thiết lập

- ❖ Phương thức thiết lập hay còn gọi là constructor, là một loại phương thức đặc biệt dùng để khởi tạo thể hiện của lớp.
- ❖ Bất kỳ một đối tượng nào được khai báo đều phải sử dụng một hàm thiết lập để khởi tạo các giá trị thành phần của đối tượng.
- ❖ Hàm thiết lập được khai báo giống như một phương thức với tên phương thức trùng với tên lớp và không có giá trị trả về (kể cả void).
- ❖ Constructor thường phải có thuộc tính public

# Phương thức thiết lập

- ❖ Constructor có thể được khai báo chồng như các hàm C++ thông thường khác hay không?
- ❖ Constructor có thể được khai báo với các tham số có giá trị ngầm định hay không?

# Ví dụ

```
class point{  
    /*Khai báo các thành phần dữ liệu*/  
    int x, y;  
    public:  
        point() { x = 0; y = 0; } /*Hàm thiết lập mặc định*/  
        point(int ox, int oy) { x = ox; y = oy; } /*Hàm thiết lập*/  
        void move (int dx, int dy);  
        void display();  
};  
point a(5,2);  
point b;  
point c(3); ?
```

# Ví dụ

```
class point{  
    /*Khai báo các thành phần dữ liệu*/  
    int x, y;  
    public:  
        point() { x = 0; y = 0; } /*Hàm thiết lập mặc định*/  
        point(int ox, int oy = 1){ x = ox; y = oy;} /*Hàm thiết lập*/  
        void move (int dx, int dy);  
        void display();  
};  
point a(5,2);  
point b;  
point c(3);
```



# Phương thức thiết lập mặc định

- ❖ Constructor mặc định (default constructor) là constructor được gọi khi thể hiện được khai báo mà không có đối số nào được cung cấp
  - `MyClass x;`
  - `MyClass* p = new MyClass;`
- ❖ Ngược lại, nếu tham số được cung cấp tại khai báo thể hiện, trình biên dịch sẽ gọi constructor khác (overload)
  - `MyClass x(5);`
  - `MyClass* p = new MyClass(5);`

# Phương thức thiết lập mặc định

- ❖ Đối với constructor mặc định, nếu ta không cung cấp bất kỳ constructor nào, C++ sẽ tự sinh constructor mặc định là một phương thức rỗng.
- ❖ Tuy nhiên, nếu ta không định nghĩa constructor mặc định nhưng lại có các constructor khác, trình biên dịch sẽ báo lỗi không tìm thấy constructor mặc định nếu ta không cung cấp tham số khi tạo thể hiện.

# Ví dụ

```
class point{  
    /*Khai báo các thành phần dữ liệu*/  
    int x, y;  
    public:  
        point(int ox, int oy = 1){ x = ox; y = oy;}  
        void move (int dx, int dy);  
        void display();  
};  
point a(5,2);  
point b;  
point c(3);
```



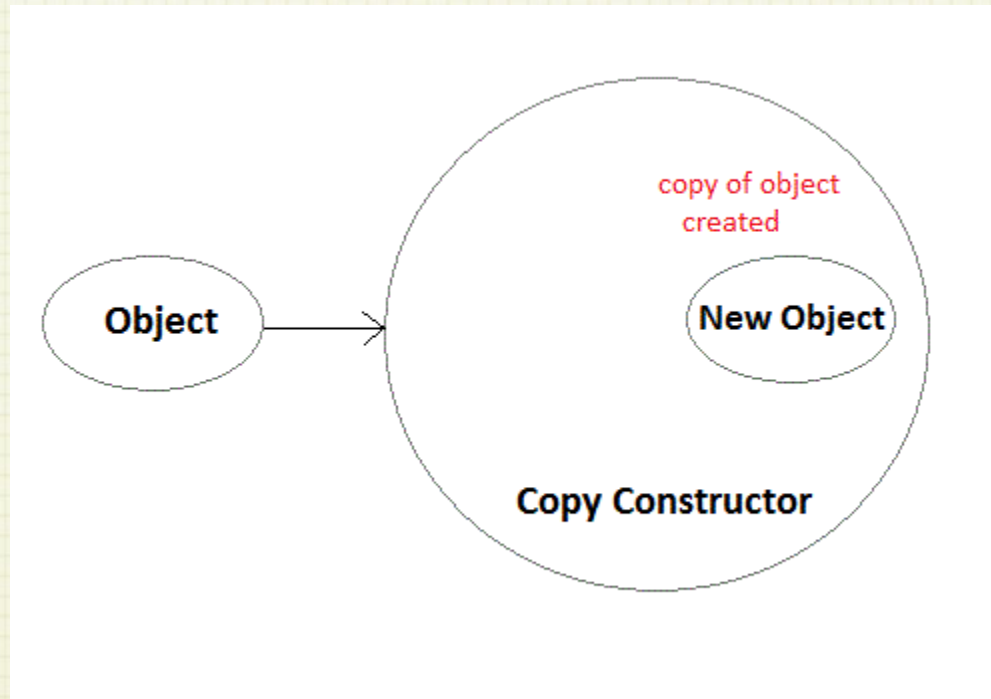
# Phương thức thiết lập sao chép

- ❖ Chúng ta có thể **tạo đối tượng mới giống đối tượng cũ** một số đặc điểm, không phải hoàn toàn như phép gán bình thường, hình thức “giống nhau” được định nghĩa theo quan niệm của người lập trình. Để làm được vấn đề này, trong các ngôn ngữ OOP cho phép ta xây dựng **phương thức thiết lập sao chép**.
- ❖ Đây là phương thức thiết lập có tham số là **hằng tham chiếu** đến đối tượng thuộc chính lớp này.



# Phương thức thiết lập mặc định

- ❖ Trong phương thức thiết lập sao chép có thể ta chỉ sử dụng một số thành phần nào đó của đối tượng ta tham chiếu → “gần giống nhau”



# Phương thức hủy bỏ

- ❖ Phương thức hủy bỏ hay còn gọi là destructor, được gọi ngay trước khi một đối tượng bị thu hồi.
- ❖ Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.



# Phương thức hủy bỏ

- ❖ Một lớp chỉ có duy nhất một **Destructor**
- ❖ Cú pháp:
  - ❖ Phương thức **Destructor** có tên trùng tên với tên lớp và có dấu ~ đặt trước
  - ❖ Không có giá trị trả về
- ❖ Được **tự động gọi** thực hiện khi đối tượng hết phạm vi sử dụng.
- ❖ Destructor phải có thuộc tính **public**

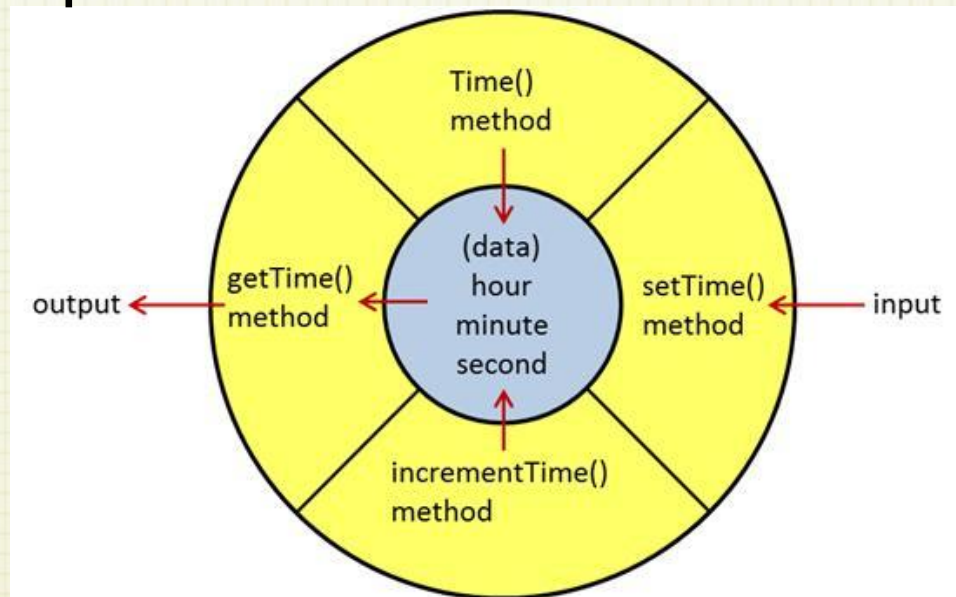
# Ví dụ

```
class vector{  
    int n;        //số chiều  
    float *v;     //vùng nhớ tọa độ  
public:  
    vector();     //Hàm thiết lập không tham số  
    vector(int size); //Hàm thiết lập một tham số  
    vector(int size, float *a);  
    ~vector();    //Hàm hủy bỏ, luôn luôn không có tham số  
    void display();  
};
```



# Thao tác với dữ liệu private

- ❖ Khi muốn **truy xuất dữ liệu private** từ các đối tượng thì phải làm thế nào?
- ❖ Khi muốn **cập nhật dữ liệu private** từ các đối tượng thì phải làm thế nào?



# Phương thức Truy vấn

- ❖ Có nhiều loại câu hỏi truy vấn có thể:
  - Truy vấn đơn giản (“giá trị của  $x$  là bao nhiêu?”)
  - Truy vấn điều kiện (“thành viên  $x$  có  $> 10$  không?”)
  - Truy vấn dẫn xuất (“tổng giá trị của các thành viên  $x$  và  $y$  là bao nhiêu?”)
- ❖ Đặc điểm quan trọng của phương thức truy vấn là nó không nên thay đổi trạng thái hiện tại của đối tượng

# Phương thức Truy vấn

- ❖ Đối với các truy vấn đơn giản, quy ước đặt tên phương thức như sau: **Tiền tố “get”**, tiếp theo là **tên của thành viên** cần truy vấn
  - `int getX();`
  - `int getSize();`
- ❖ Các loại truy vấn khác nên có tên có tính mô tả
- ❖ Truy vấn điều kiện nên có **tiền tố “is”**



# Phương thức Cập nhật

- ❖ Thường để thay đổi trạng thái của đối tượng bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó.
- ❖ Dạng đơn giản nhất là gán một giá trị nào đó cho một thành viên dữ liệu.
- ❖ Đối với dạng cập nhật đơn giản, quy ước đặt tên như sau: Dùng tiền tố “set” kèm theo tên thành viên cần sửa
  - `int setX(int);`



# Truy vấn và Cập nhật

- ❖ Nếu **phương thức get/set** chỉ có nhiệm vụ cho ta đọc/ghi giá trị cho các thành viên dữ liệu → Quy định các thành viên **private** để được ích lợi gì?
  - Ngoài việc **bảo vệ các nguyên tắc đóng gói**, ta cần **kiểm tra xem giá trị mới** cho thành viên dữ liệu có hợp lệ hay không.
  - Sử dụng phương thức truy vấn cho phép ta thực hiện việc **kiểm tra trước khi thực sự thay đổi giá trị** của thành viên.
  - Chỉ cho phép các dữ liệu có thể truy vấn hay thay đổi mới được truy cập đến.

# Ví dụ

```
int Student::setGPA (double newGPA){  
    if ((newGPA >= 0.0) && (newGPA <= 4.0)){  
        this->gpa = newGPA;  
        return 0; // Return 0 to indicate success  
    }  
    else  
    {  
        return -1; // Return -1 to indicate failure  
    }  
}
```

# Thành viên tĩnh – static member

- ❖ Trong C, **static** xuất hiện trước dữ liệu được khai báo trong một hàm nào đó thì giá trị của dữ liệu đó vẫn được lưu lại như một biến toàn cục.
- ❖ Trong C++, nếu **static** xuất hiện trước một dữ liệu hoặc một phương thức **của lớp** thì giá trị của nó vẫn được lưu lại và **có ý nghĩa cho đối tượng khác của cùng lớp này**.
- ❖ Các thành viên **static** có thể là **public**, **private** hoặc **protected**.



# Thành viên tĩnh – static member

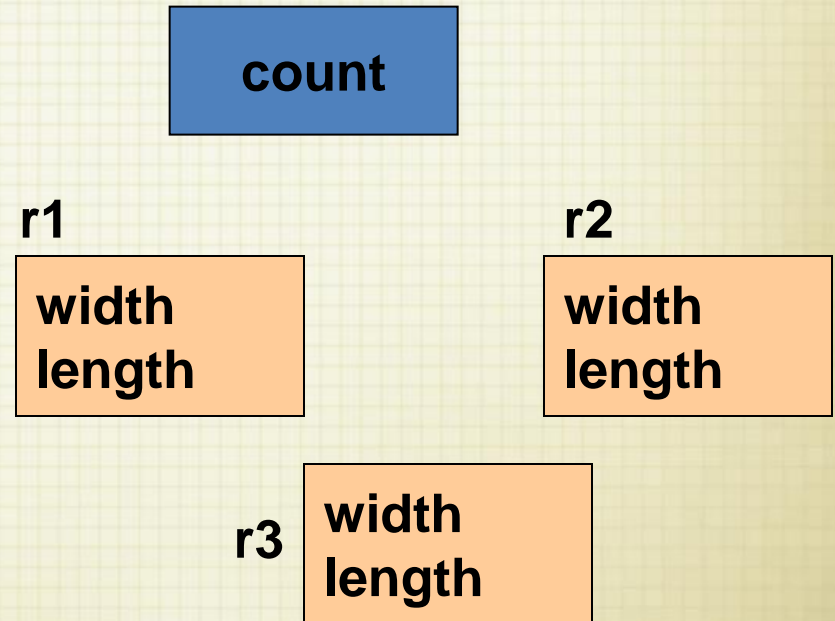
- ❖ Đối với class, **static** dùng để **khai báo thành viên dữ liệu dùng chung cho mọi thể hiện của lớp**:
  - Một bản duy nhất tồn tại trong suốt quá trình chạy của chương trình.
  - Dùng chung cho tất cả các thể hiện của lớp.
  - Bất kể lớp đó có bao nhiêu thể hiện.



# Ví dụ

```
class Rectangle
{
    private:
        int width;
        → int length;
        static int count;
    public:
        void set(int w, int l);
        int area();
}
```

```
Rectangle r1;
Rectangle r2;
Rectangle r3;
```



# Ví dụ

❖ Đếm số đối tượng MyClass:

```
class MyClass{  
    public:  
        MyClass();  
        ~MyClass();  
        void printCount();  
    private:  
        static int count;  
};
```

# Ví dụ

```
int MyClass::count = 0;
MyClass::MyClass(){
    this → count++;
}
MyClass::~MyClass(){
    this → count--;
}
void MyClass::printCount(){
    cout << "There are currently " << this → count << "
instance(s) of MyClass.\n";
}
```

# Ví dụ

```
void main()
{
    MyClass* x = new MyClass;
    x → printCount();
    MyClass* y = new MyClass;
    x → printCount();
    y → printCount();
    delete x;
    y → printCount();
}
```



# Thành viên tĩnh – static member

## ❖ Phương thức static?

- Đối với các phương thức static, ngoài ý nghĩa tương tự với dữ liệu, còn có sự khác biệt cơ bản đó là việc cho phép truy cập đến các phương thức static khi **chưa khai báo đối tượng** (thông qua tên lớp)

# Thành viên tĩnh – static member

- ❖ Các thành viên lớp tĩnh public có thể được truy cập thông qua bất kỳ đối tượng nào của lớp đó, hoặc chúng có thể được truy cập thông qua tên lớp sử dụng toán tử định phạm vi.
- ❖ Các thành viên lớp tĩnh private và protected phải được truy cập thông qua các hàm thành viên public của lớp hoặc thông qua các friend của lớp.
- ❖ Các thành viên lớp tĩnh tồn tại ngay cả khi đối tượng của lớp đó không tồn tại.

# Thành viên tĩnh – static member

- ❖ Để truy cập một thành viên lớp tĩnh public khi các đối tượng của lớp không tồn tại, đơn giản thêm vào đầu tên lớp và toán tử định phạm vi cho thành viên dữ liệu.
- ❖ Để truy cập một thành viên lớp tĩnh private hoặc protected khi các đối tượng của lớp không tồn tại, một hàm thành viên public phải được cung cấp và hàm phải được gọi bởi thêm vào đầu tên của nó với tên lớp và toán tử định phạm vi.



# Ví dụ về đối tượng toàn cục

- ❖ Xét đoạn chương trình sau:

```
#include <iostream.h>
void main(){
    cout << "Hello, world.\n";
}
```

- ❖ Hãy sửa lại đoạn chương trình trên để có kết xuất:

Entering a C++ program saying...

Hello, world.

And then exiting...

- ❖ Yêu cầu không thay đổi hàm main() dưới bất kỳ hình thức nào.



# Ví dụ về đối tượng toàn cục

```
#include <iostream.h>

class Dummy{
public:
    Dummy(){cout << "Entering a C++ program saying...\n";}
    ~Dummy(){cout << "And then exiting...";}
};

Dummy A;

void main(){
    cout << "Hello, world.\n";
}
```

# Bài tập (20 phút)

Định nghĩa lớp **cPhanSo** biểu diễn khái niệm phân số gồm 2 thành phần dữ liệu: **tử số**, **mẫu số** và các thao tác:

- Nhập phân số
- Khởi tạo tử số và mẫu số cho phân số
- Xuất phân số
- Lấy giá trị phân số
- Tính tổng 2 phân số

Viết chương trình cho phép người dùng nhập vào 2 phân số, tính tổng 2 phân số và xuất kết quả ra màn hình dưới dạng phân số.

# Bài tập

1. Viết chương trình nhập họ tên, điểm toán, điểm văn của một học sinh. Tính điểm trung bình, xếp loại và xuất kết quả.
2. Viết chương trình nhập tọa độ tâm và bán kính của đường tròn. Tính diện tích và chu vi của đường tròn.
3. Viết chương trình nhập vào tọa độ 2 điểm trong mặt phẳng Oxy. Tính khoảng cách giữa chúng và xuất kết quả.



# Bài tập

4. Cài đặt lớp cArray để biểu diễn mảng một chiều các số nguyên cho phép thực hiện các yêu cầu sau:
- Tạo lập mảng gồm n số nguyên ngẫu nhiên bằng constructor
  - Xuất mảng ra màn hình
  - Tìm số âm lớn nhất
  - Đếm lần xuất hiện của một số nguyên x
  - Kiểm tra mảng có giảm dần không
  - Sắp xếp mảng tăng dần



# Bài tập

5. Định nghĩa lớp `cHocSinh` gồm các thuộc tính: mã, họ tên, giới tính, năm sinh, điểm trung bình và các phương thức cần thiết.

Viết chương trình cho phép người dùng nhập thông tin 2 học sinh.

- Cho biết học sinh nào có điểm trung bình cao hơn.
- Cho biết học sinh nào có tuổi nhỏ hơn

# Bài tập

6. Cài đặt lớp số phức để biểu diễn khái niệm số phức (một số phức bao gồm hai thành phần: phần ảo, phần thực).

Cho phép thực hiện các thao tác sau:

- Tạo lập số phức khi biết phần thực và ảo
- Thay đổi phần thực, phần ảo
- Lấy giá trị phần thực, phần ảo
- Nhập số phức
- Xuất số phức
- Công hai số phức.

❖ Viết chương trình minh họa các chức năng của lớp số phức

Xét hai số phức  $A(a_1, a_2)$ ,  $B(b_1, b_2)$

- $A + B = (a_1 + b_1, a_2 + b_2)$

- $A - B = (a_1 - b_1, a_2 - b_2)$

- $A * B = (a_1 * b_1 - a_2 * b_2, a_1 * b_2 + a_2 * b_1)$

- $A / B = \left( \frac{a_1 * b_1 + a_2 * b_2}{b_1^2 + b_2^2}, \frac{b_1 * a_2 - a_1 * b_2}{b_1^2 + b_2^2} \right)$

# BT Cộng điểm

**23521016@gm.uit.edu.vn**