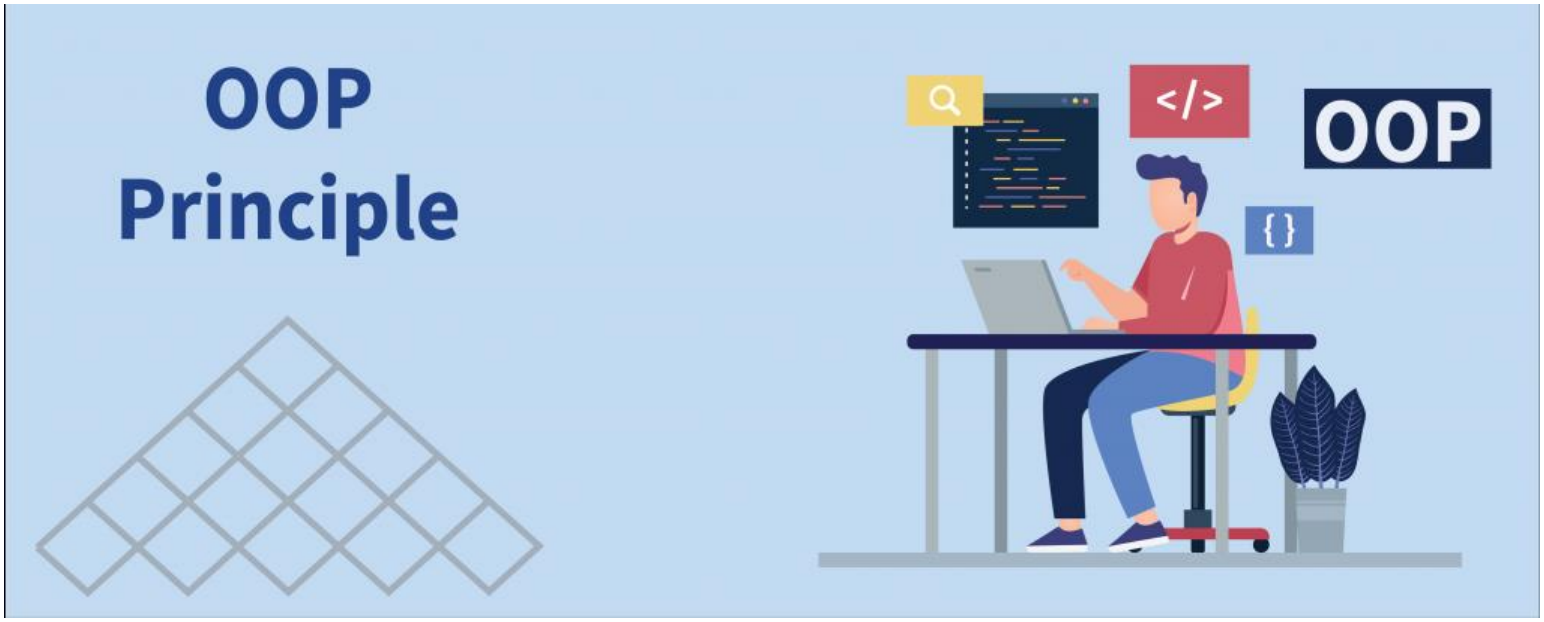


LỚP CƠ BẢN

Ths. Đinh Thị Thúy – AI LAB/TLU
Thuydt@thanglong.edu.vn

NỘI DUNG

- Tạo lớp
- Tạo đối tượng
- Hàm tạo & hàm huỷ
- Mảng đối tượng

Đọc:

- Starting out with C++ Ch. 14.

TẠO LỚP

- Khai báo lớp
  - Cấu trúc lớp
  - Từ khoá public, private, const
  - Setter và getter
- Định nghĩa lớp

KHAI BÁO LỚP

class *<tên lớp>*

{

<khai báo biến>;

<khai báo nguyên mẫu hàm>;

};

Dấu ; sau khai báo lớp

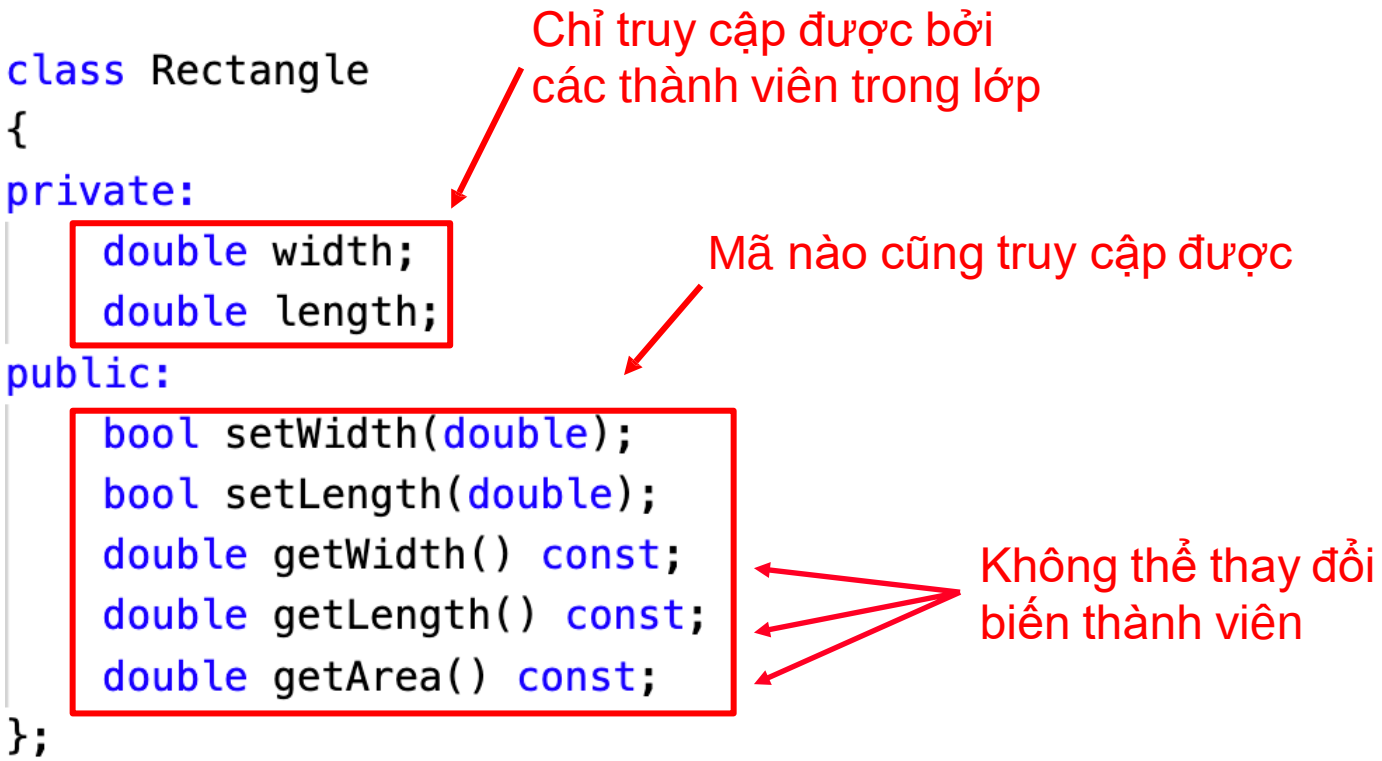
```
class Rectangle
{
private :
    double width;
    double length;
public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const ;
    double getLength()const ;
    double getArea()  const;
};
```

Từ khoá kiểm soát truy cập

Từ khoá const

- Thành viên private chỉ có thể truy cập từ các thành viên cùng lớp
- Thành viên public có thể truy cập từ mọi nơi (thuộc lẫn không thuộc) lớp
- Cú pháp: <private/public> :<các khai báo thành viên>
- Cú pháp <khai báo hàm> const sẽ cấm hàm thay đổi biến thành viên
- Nếu hàm có lệnh thay đổi biến (vd phép gán) sẽ báo lỗi

VD: PUBLIC, PRIVATE,CONST




```
class Rectangle
{
private:
    double width;
    double length;
public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};
```

TỪ KHÓA PUBLIC, PRIVATE (TIẾP)

- public, private có thể xuất hiện nhiều lần, theo bất kỳ thứ tự nào
- Nếu không chỉ định thì mặc định là private
- Các biến luôn private, các hàm nói chung public
  - Trừ hàm tiện ích nên để private (~học sau)

```
class Rectangle
{
private:
    double width;
    double length;
public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};
```

```
class Rectangle
{
    double length;
public:
    bool setLength(double);
    bool setWidth(double);
private:
    double width;
public:
    double getLength() const;
    double getWidth() const;
    double getArea() const;
};
```

```
class Rectangle
{
    double width;
    double length;
public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};
```

SETTER & GETTER (HÀM TRUY CẬP)

- Biến private vẫn truy cập gián tiếp qua các hàm thành viên public
- Setter (Hàm truy cập ghi): cập nhật giá trị một/nhiều biến thành viên
- Getter (Hàm truy cập đọc): lấy giá trị từ biến thành viên (để const ??)

```
class Rectangle
{
    double width;
    double length;
public:
    bool setWidth(double);
    bool setLength(double);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};
```

Biến luôn private (mặc định)

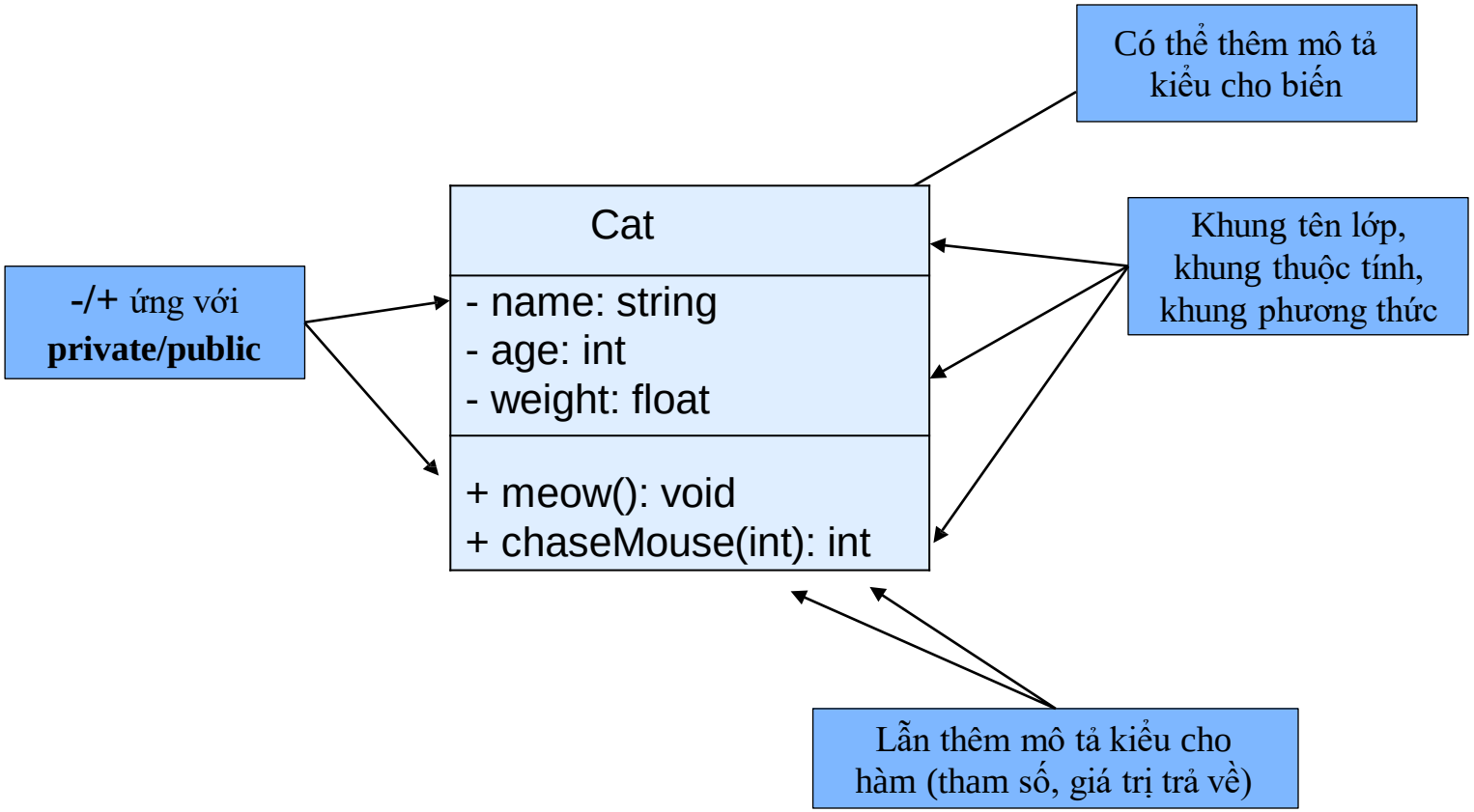
Hàm truy cập luôn public

Không phải hàm truy cập song vẫn trả giá trị

- Có thể viết nhiều hoặc chỉ một setter chung:
  - setName, setAge, setWeight
  - hoặc setInfor cập nhật cả 3 biến
  - *Đi thi làm như đề, chỉ thêm hàm khi cần*

Cat
name age weight
meow() chaseMouse()

CÚ PHÁP UML MÔ TẢ LỚP



- Bài tập: Viết khai báo lớp cho Cat, tự thêm đủ setter, getter cho từng biến, tự đặt private, public, const phù hợp

VD: KHAI BÁO LỚP CAT

```
class Cat{
    string name;
    int age;
    float weight;
public:
    void setName(string);
    void setAge (int);
    void setWeight(float);
    string getName() const;
    int getAge() const;
    float getWeight() const;
    void meow() const;
    int chaseMouse(int) const;
};
```

HAI CÁCH ĐỊNH NGHĨA HÀM THÀNH VIÊN

- Định nghĩa trong khai báo: thay luôn nguyên mẫu bằng định nghĩa :

```
void setWidth(double);      void setWidth(double w) {width = w;}
double getWidth() const;    double getWidth() const {return width;}
```

  - Hàm thành viên truy cập biến thành viên như thể biến đó khai báo trong hàm
- Định nghĩa ngoài khai báo (sau khi đã khai báo)
  - cần nêu rõ hàm lớp nào bằng toán tử phân giải phạm vi (::)

```
kiểu_trả_về tên_lớp :: tên_hàm (ds_tham_số) [const] {
        mã_thân_hàm
 }
```

  - const nếu xuất hiện ở nguyên mẫu thì phải xuất hiện ở định nghĩa

```
void Rectangle::setLength(double l){length = l;}
double Rectangle::getLength() const {return length; }
```

VD: ĐỊNH NGHĨA TRONG LỚP RECTANGLE

```
class Rectangle
{
    double width;
    double length;
public:
    void setWidth(double w){ width = w; }
    void setLength(double l){ length = l; }
    double getWidth() const { return width; }
    double getLength() const { return length; }
    double getArea() const { return width * length; };
};
```

VD: ĐỊNH NGHĨA NGOÀI LỚP RECTANGLE

```
class Rectangle
{
    double width;
    double length;
public:
    void setWidth(double w);
    void setLength(double l);
    double getWidth() const;
    double getLength() const;
    double getArea() const;
};


void Rectangle::setWidth(double w){ width = w; }
void Rectangle::setLength(double l){ length = l; }
double Rectangle::getWidth() const { return width; }
double Rectangle::getLength() const { return length; }
double Rectangle::getArea() const { return width * length; }
```

VD: ĐỊNH NGHĨA LỚP CAT

- setInfor cập nhật cả name, age, weight
- 3 getter cho name, age, weight
- meow() in ra “Meow, I am ” + name
- chaseMouse() nhập vào số chuột cần bắt, trả về số chuột bắt được, biết:
  - Với mỗi chuột, sinh ngẫu nhiên số 0/1
  - Nếu ra 0 là không bắt được con chuột đó, ra 1 là bắt được
- Định nghĩa các hàm setInfor, getName, getAge, getWeight trong lớp
- Định nghĩa các hàm meow, chaseMouse ngoài lớp

Cat
- name: string - age: int - weight: float
+ setInfor(string, int, float): void + getName() const: string + getAge() const: int + getWeight() const: float + meow(): void + chaseMouse(int): int

VD: ĐỊNH NGHĨA HÀM TRONG LỚP

```
class Cat{
    string name;
    int age;
    float weight;
public:
    void setInfor(string n, int a, float w){
        name = n;
        age = a;
        weight = w;
    }
    string getName() const { return name; }
    int getAge() const { return age; }
    float getWeight() const { return weight; }
    void meow() const;
    int chaseMouse(int) const;
};
```

VD: ĐỊNH NGHĨA HÀM BÊN NGOÀI LỚP

```
void Cat::meow() const { cout << "Meow, I am " << name; }
int Cat::chaseMouse(int n) const {
    srand(time(0));
    int count = 0;
    for (int i = 0; i<n; i++){
        int tmp = rand() % 2;
        if (tmp) count++;
    }
    return count;
}
```

TẠO ĐỐI TƯỢNG

- Khai báo đối tượng, truy cập thành viên
- Khởi tạo thuộc tính trên dòng lệnh

KHAI BÁO ĐỐI TƯỢNG VÀ TRUY CẬP THÀNH VIÊN

- Khai báo như khai báo biến:

```
Rectangle r; // coi Rectangle là một kiểu
```

- Truy cập các thành viên sử dụng toán tử dấu chấm (.) :
  - hàm không thuộc lớp chỉ truy cập được thành viên public

```
r.width = 5.5; // error
r.setWidth(5.2);
cout << r.getWidth() << endl;
```

```
5.2
```

- Có thể đọc biến thành viên trước cả khi ghi nó không?

```
Rectangle r;
cout << r.getWidth() << endl;
```

```
0
```

  - Tuỳ phiên bán trình biên dịch mà biến thành viên được khởi tạo mặc định bởi null hoặc giá trị ngẫu nhiên

VD: TRUY CẬP KHI CHƯA KHỞI TẠO THÀNH VIÊN

```
class Rectangle
{
    double width;
    double length;
public:
    void setWidth(double w){ width = w; }
    void setLength(double l){ length = l; }
    double getWidth() const { return width; }
    double getLength() const { return length; }
    double getArea() const { return width * length; };
};

int main(){
    Rectangle r;
    cout << r.getWidth() << " " << r.getLength() << " " << r.getArea() << endl;
    r.setWidth (5);
    r.setLength (12);
    cout << r.getWidth() << " " << r.getLength() << " " << r.getArea() << endl;
    return 0;
}
```

```
0 6.95312e-310 0
5 12 60
```

KHỞI TẠO BIẾN THÀNH VIÊN TỪ DÒNG LỆNH

- Để giá trị ban đầu của biến luôn xác định và khác null, hãy khởi tạo sẵn chúng:

```
class Rectangle
{
    double width = 2;
    double length = 5;
```

```
Rectangle r;
cout << r.getWidth() << " " << r.getLength();
```

```
2 5
```

- Để cấm biến thành viên thay đổi, để biến là hằng:

```
class Rectangle
{
    const int myConst = 10;
```

  - mọi đối tượng đều có myConst = 10, biến thành viên vô nghĩa?
  - => cần cách linh hoạt hơn để khởi tạo biến: hàm tạo (~bài kế)

VD: KHỞI TẠO BIẾN THÀNH VIÊN

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:
    void setWidth(double w){ width = w; }
    void setLength(double l){ length = l; }
    double getWidth() const { return width; }
    double getLength() const { return length; }
    double getArea() const { return width * length; };
};

int main(){
    Rectangle r;
    cout << r.getWidth() << " " << r.getLength() << " " << r.getArea() << endl;
    r.setWidth (5);
    r.setLength (12);
    cout << r.getWidth() << " " << r.getLength() << " " << r.getArea() << endl;
    return 0;
}
```

```
2 5 10
5 12 60
```

VD: CHƯƠNG TRÌNH LỚP CAT

- Khởi tạo biến thành viên:
  - Sửa lại định nghĩa lớp sao cho giả sử khi sinh ra mọi con mèo đều mặc định tên “Bibi”, 1 tuổi và nặng 2 kg
- Khai báo đối tượng, truy cập thành viên:
  - Tạo một con mèo, in thông tin, sau đó thay đổi tên, tuổi, cân nặng, in ra thông tin mới, cho con mèo kêu, và bảo nó bắt một số chuột nhập từ bàn phím, in ra số chuột mèo bắt được

```
class Cat{
    string name = "Bibi";
    int age = 1;
    float weight = 2;
int main(){
    Cat c;
    cout << c.getName() << " " << c.getAge() << " " << c.getWeight() << endl;
    c.setInfor ("Tom", 2, 3);
    cout << c.getName() << " " << c.getAge() << " " << c.getWeight() << endl;
    c.meow();
    int n;
    cout << "so chuot: "; cin >> n;
    cout << c.chaseMouse(n) << endl;
    return 0;
}
```

HÀM TẠO VÀ HÀM HUỶ

- Khái niệm hàm tạo
- Các loại hàm tạo nạp chồng
- Khái niệm hàm huỷ

HÀM TẠO (CẤU TỬ - CONSTRUCTOR)

- Là hàm thành viên được C++ tự động gọi khi cần tạo đối tượng
- Cú pháp: tên_lớp (danh_sách_tham_số){ thân_hàm }

```
Rectangle(…){…}
```

  - Tên hàm trùng tên lớp
  - Không có kiểu trả về
  - Được phép có nhiều hàm tạo miễn danh sách tham số khác nhau (??)
- Hàm tạo nên chứa mã khởi tạo các biến thành viên

HÀM TẠO MẶC ĐỊNH

- Là hàm tạo được gọi để tạo đối tượng mà không cần đối số

```
Rectangle r; // gọi hàm tạo mặc định, nếu không có sẽ báo lỗi
```

- Nên có, kể cả khi nó rỗng (??). Nó có thể tạo bằng 3 cách:
  1. Viết hàm tạo không tham số
  2. Viết hàm tạo mà mọi tham số đều có đối số mặc định
  3. Không viết hàm tạo nào, C ++ sẽ tự sinh một hàm tạo rỗng tham số (nên là hàm tạo mặc định) và rỗng thân (không làm gì cả).
     - Đã viết ít nhất 1 hàm tạo thì C++ không cho thêm hàm nào
- Một lớp có nhiều nhất 1 hàm tạo mặc định (??)

VD: HÀM TẠO MẶC ĐỊNH (CÁCH 1)

- Viết hàm tạo mặc định theo kiểu không tham số cho lớp Rectangle, để khi khai báo một Rectangle không truyền đối số thì ta sẽ có chiều rộng là 2, chiều dài là 5.

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:

Rectangle(){}

Rectangle(): width(2), length(5) {}
```

```
class Rectangle
{
    double width;
    double length;
public:
    Rectangle(): width(2), length(5) {}
};
```

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:
    Rectangle(): width(3), length(10) {}
}
```

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:
    Rectangle(): length(10) {}
}
```

Có những tham số chỉ khởi tạo được bằng danh sách, VD: hằng, biến tham chiếu

VD: HÀM TẠO MẶC ĐỊNH (CÁCH 1)

- Viết hàm tạo mặc định theo kiểu không tham số cho lớp Rectangle, để khi khai báo một Rectangle không truyền đối số thì ta sẽ có chiều rộng là 2, chiều dài là 5.

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:
Rectangle() {
    setWidth(2);
    setLength(5);
}
Rectangle() {
    setInfor(2,5);
}
void setInfor(double w, double l) {
    width = w;
    length = l;
}
```

```
Rectangle() {
    setInfor(2,5);
}
void setInfor(double w, double l) {
    if (width < 0)  throw "Loi\n";
    width = w;
    if (length < 0)  throw "Loi\n";
    length = l;
}

Rectangle() {
    width = 2;
    length = 5;
}
```

Nếu có setter tổng, cách dễ nhất viết hàm tạo là gọi setter này

VD: HÀM TẠO MẶC ĐỊNH (CÁCH 1)

- Viết hàm tạo mặc định theo kiểu không tham số cho lớp Cat, để khi khai báo đối tượng không truyền đối số ta có một con mèo tên Bibi, 1 tuổi, nặng 2kg. Thử làm bằng 3 cách khác nhau

```
class Cat{
    string name = "Bibi";
    int age = 1;
    float weight = 2;
public:
    Cat(){}
```

```
Cat(){ setInfor("Bibi", 1, 2); }
```

```
Cat(): name("Bibi"), age(1), weight(2){}
```

```
Cat(){
    name = "Bibi";
    age = 1;
    weight = 2;
}
```

HÀM TẠO CÓ THAM SỐ, CÓ TRUYỀN ĐỐI SỐ

- Giống hàm có tham số bình thường: coi tham số là biến trong hàm

```
Rectangle(double w, double l){
    width = w;
    length = l;
}
```

```
Rectangle::Rectangle(double w, double l){
    width = w;
    length = l;
}
```

- Hàm tạo sẽ được truyền đối số trên dòng khai báo đối tượng:

```
Rectangle r(2,10);
cout << r.getWidth() << " " << r.getLength() << endl;
```

```
2 10
```

- Tuy nhiên, nếu mọi tham số hàm tạo đều có đối số mặc định thì có thể không truyền đối số => biến thành hàm tạo mặc định

VD: HÀM TẠO CÓ THAM SỐ, CÓ TRUYỀN ĐỐI SỐ

- Viết thêm các hàm tạo sau cho lớp Rectangle:
  - có 2 tham số khởi tạo width, length
  - có 1 tham số khởi tạo width; còn length gán bằng 5

```
class Rectangle
{
    double width = 2;
    double length = 5;
public:
    Rectangle(){}
    Rectangle(double w): width(w){}
    // Rectangle(double width): width(width){}
    Rectangle(double w, double l){
        width = w;
        length = l;
    }
```

VD: HÀM TẠO CÓ THAM SỐ, CÓ TRUYỀN ĐỐI SỐ

- Viết thêm các hàm tạo sau cho lớp Cat:
  - có 1 tham số khởi tạo name, còn age, weight gán bằng 1 và 2
  - có 2 tham số khởi tạo name, age; còn weight gán bằng 2
  - có 3 tham số khởi tạo name, age, weight
- Cách 1:

```
Cat(){ setInfor("Bibi", 1, 2); }
Cat(string name){ setInfor(name, 1, 2); }
Cat(string name, int age){ setInfor(name, age, 2); }
Cat(string name, int age, float weight){ setInfor(name, age, weight); }
```

- Cách 2:

```
string name = "Bibi";
int age = 1;
float weight = 2;
Cat(){}
Cat(string name): name(name){}
Cat(string name, int age): name(name), age(age){}
Cat(string name, int age, float weight): name(name), age(age), weight(weight){}
```

VD: HÀM TẠO CÓ THAM SỐ, CÓ TRUYỀN ĐỐI SỐ

```
Cat(){ setInfor("Bibi", 1, 2); }
Cat(string name){ setInfor(name, 1, 2); }
Cat(string name, int age){ setInfor(name, age, 2); }
Cat(string name, int age, float weight){ setInfor(name, age, weight); }
```

```
int main(){
    cout << endl;
    Cat c1, c2 ("Tom"), c3 ("Tim", 2), c4("Toto", 3, 5);
    cout << c1.getName() << " " << c1.getAge() << " " << c1.getWeight() << endl;
    cout << c2.getName() << " " << c2.getAge() << " " << c2.getWeight() << endl;
    cout << c3.getName() << " " << c3.getAge() << " " << c3.getWeight() << endl;
    cout << c4.getName() << " " << c4.getAge() << " " << c4.getWeight() << endl;
    cout << endl;
    return 0;
}
```

```
Bibi 1 2
Tom 1 2
Tim 2 2
Toto 3 5
```

ÔN LẠI: ĐỐI SỐ MẶC ĐỊNH

- Là đối số tự động truyền nếu lời gọi hàm bỏ qua vị trí đó
  - void evenOrOdd(int = 0); // có nguyên mẫu thì đặt đối số mặc định
  - void evenOrOdd(int a){…} // định ở nguyên mẫu và bỏ ở tiêu đề

    float square(float a = 0){…} // chỉ để tiêu đề nếu bỏ nguyên mẫu
- Được quyền có đối số mặc định cho nhiều tham số miễn tham số có đối mặc định không được ở trước tham số không có
  - int getSum(int, int = 0, int = 0); // OK
  - int getSum(int, int = 0, int); // báo lỗi
- Lời gọi hàm được bỏ qua các đối số đã khai mặc định: getSum(a);

  getSum(a, b); getSum(a, b, c); // 3 lời gọi đều OK

HÀM TẠO MẶC ĐỊNH CÁCH 2: KHI MỌI ĐỐI SỐ HÀM ĐỀU MẶC ĐỊNH

Lớp Rectangle: Viết hàm tạo mặc định theo kiểu hàm tạo có tham số nhưng mọi đối số đều mặc định

```
class Rectangle
{
    double width;
    double length;
public:
    Rectangle(double w = 2, double l = 5): width(w), length(l){}

int main(){
    cout << endl;
    Rectangle r1, r2(7), r3(3, 9);
    cout << r1.getWidth() << " " << r1.getLength() << endl
         << r2.getWidth() << " " << r2.getLength() << endl
         << r3.getWidth() << " " << r3.getLength() << endl;
    return 0;
}
```

```
2 5
7 5
3 9
```

VD: HÀM TẠO CÓ THAM SỐ, KHÔNG CẦN ĐỐI SỐ

- Lớp Cat: viết lại hàm tạo mặc định theo kiểu có tham số nhưng mọi đối số đều mặc định
- Có nhận xét gì về hàm mới viết và các hàm tạo cũ?

```
Cat(string name = "Bibi", int age = 1, float weight = 2)
{ setInfor(name, age, weight); }
```

```
Bibi 1 2
Tom 1 2
Tim 2 2
Toto 3 5
```

```
int main(){
    cout << endl;
    Cat c1, c2 ("Tom"), c3 ("Tim", 2), c4("Toto", 3, 5);
    cout << c1.getName() << " " << c1.getAge() << " " << c1.getWeight() << endl;
    cout << c2.getName() << " " << c2.getAge() << " " << c2.getWeight() << endl;
    cout << c3.getName() << " " << c3.getAge() << " " << c3.getWeight() << endl;
    cout << c4.getName() << " " << c4.getAge() << " " << c4.getWeight() << endl;
    cout << endl;
    return 0;
}
```

TỔNG KẾT VỀ NẠP CHỒNG HÀM TẠO

- Có thể nạp chồng hàm tạo miễn danh sách tham số khác nhau.
- Tuỳ danh sách đối số khi khai báo biến, C++ sẽ gọi hàm tạo phù hợp
- Nhiều nhất một hàm tạo mặc định, nhiều hơn sẽ báo lỗi.

```
Square();
Square(int = 0);  // lỗi biên dịch (??)
```

- Ngược lại nếu không có hàm tạo mặc định thì khai báo biến phải truyền đối số ứng với một hàm tạo đã có, không sẽ báo lỗi.
- Nên viết hàm tạo mặc định theo kiểu dùng đối số mặc định, khi đó cùng lúc sẽ có nhiều phiên bản gọi hàm

VD: CHƯƠNG TRÌNH LỚP EMPLOYEE

- Viết lớp Employee có các thành viên:
  - name: Một chuỗi chỉ tên của nhân viên
  - idNumber: Một biến int chỉ số ID của nhân viên
  - department: Một chuỗi chỉ bộ phận nhân viên làm việc
  - position:. Một chuỗi chỉ chức danh của nhân viên
  - Các hàm setter và getter cho TỪNG BIẾN thành viên trên
- Viết hàm main tạo ra 1 đối tượng Employee ở dòng đầu bảng (Susan Meyers), sau đó in ra thông tin đối tượng này
-

Name	ID Number	Department	Position
Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

VD: HOÀN THIỆN LỚP EMPLOYEE

- Thêm vào lớp Employee cũ các thành viên sau:
  - Hàm tạo 4 tham số gán cho name, idNumber, department, position.
  - Hàm tạo 2 tham số gán cho name và idNumber, và đặt department, position thành chuỗi rỗng
  - Hàm tạo mặc định đặt chuỗi rỗng cho name, department, position, và đặt 0 cho idNumber.
- Sửa hàm main tạo cả 3 đối tượng trong bảng rồi in ra thông tin:

Name	ID Number	Department	Position
Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

- Có nhận xét gì khi viết hàm main lần này so với lần trước?

HÀM HỦY (HUỶ TỬ)

- Là hàm thành viên tự động gọi khi một đối tượng bị phá hủy
- Cú pháp: ~tên_lớp (){ thân_hàm }

```
~Rectangle(){…}
```

  - Không kiểu trả về, không tham số
- Có duy nhất 1 hàm hủy, không thể nạp chồng. Nếu không viết thì C++ cũng tự cho một hàm huỷ mặc định thân rỗng.
- Cần viết hàm huỷ khi có biến thành viên là con trỏ cấp phát động

VD: HÀM HUỶ LỚP RECTANGLE

```
class Rectangle
{
    double width;
    double length;
public:
    Rectangle(double w = 2, double l = 5): width(w), length(l){}
    ~Rectangle(){}
    void setWidth(double w){ width = w; }
    void setLength(double l){ length = l; }
    double getWidth() const { return width; }
    double getLength() const { return length; }
    double getArea() const { return width * length; };
};

Rectangle::~Rectangle(){}

~Cat(){}

~Employee(){}
```

VD: HÀM HUỶ DỌN DẸP BỘ NHỚ

Lớp Number Array

Thiết kế một lớp có các thành viên:

- Một mảng số thực cấp phát động.
- Một hàm tạo nhận một tham số nguyên và sẽ dùng nó làm kích thước cấp phát động mảng.
- Hàm huỷ giải phóng bộ nhớ mảng.
- Hàm thiết lập giá trị cho phần tử mảng ở vị trí cho trước
- Hàm trả về phần tử mảng ở vị trí cho trước
- Hàm trả về giá trị lớn nhất mảng
- Hàm trả về giá trị nhỏ nhất mảng
- Hàm trả về giá trị trung bình của mảng

Biểu diễn lớp trong một chương trình.

VD: HÀM HUỶ DỌN DẸP BỘ NHỚ

```
class NumberArray
{
private:
    int size;
    double * a;
    inline int check (int) const;
public:
    NumberArray(int = 50);
    ~NumberArray();
    void setElement(double, int);
    double getElement (int) const;
    double getMin() const;
    double getMax() const;
    double getAverage() const;
};
```

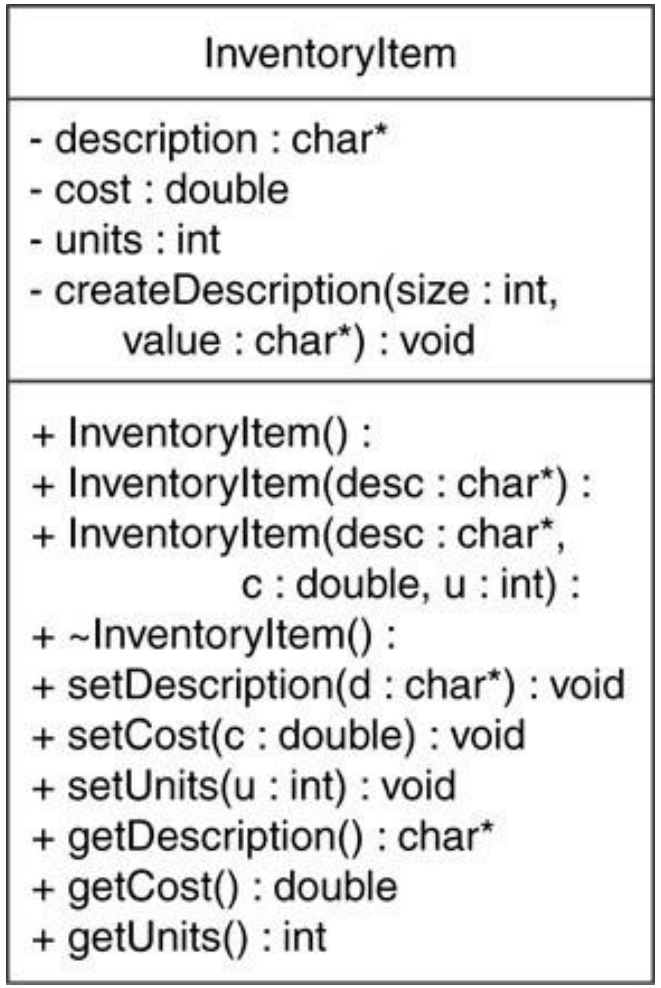
```
NumberArray(int size = 50)
: size(size), a(new double [size]){}

~NumberArray() { delete [] a; }
```

UML VỚI HÀM TẠO VÀ HÀM HỦY

Không có kiểu trả về nào được liệt kê cho các hàm tạo hoặc hàm hủy, nhưng vẫn có dấu :

Constructors

Destructor



MẢNG ĐỐI TƯỢNG

- Định nghĩa
- Khởi tạo
- Duyệt mảng

LỚP: InventoryItem

```
class InventoryItem {
private:
    string description; // mô tả mặt hàng
    double cost;        // chi phí
    int units;          // số lượng có trong kho
public:
    // hàm tạo #1
    InventoryItem()
    : description(""), cost(0), units(0) {}

    // hàm tạo #2
    InventoryItem(string desc)
    : description(desc), cost(0.0), units(0)  {}

    // hàm tạo #3
    InventoryItem(string desc, double c, int u)
    : description(desc), cost (c), units(u) {}
```

MẢNG ĐỐI TƯỢNG

- Lớp là một kiểu dữ liệu, nên có thể tạo mảng:

  InventoryItem inventory[40];

- Khai báo mà không khởi tạo sẽ gọi hàm tạo mặc định trên cả mảng
  - Một tình huống nữa bất lợi nếu không sẵn hàm tạo mặc định
- Muốn gọi hàm tạo một đối số, khởi tạo bằng danh sách sau:

  InventoryItem inventory[3] ={ "Hammer", "Wrench", "Pliers" };

  Muốn gọi hàm tạo nhiều đối số, gọi tường minh tên hàm tạo

```
InventoryItem inventory[3] = { InventoryItem("Hammer", 6.95, 12),
                               InventoryItem("Wrench", 8.75, 20),
                               InventoryItem("Pliers", 3.75, 10) };
```

(InventoryItem là lớp có 3 hàm tạo: mặc định, 1 tham số, 3 tham số)

- Có thể gọi trộn nhiều hàm tạo:

```
InventoryItem inventory[3] = { "Hammer",
                               InventoryItem("Wrench", 8.75, 20),
                               "Pliers" };
```

- Mảng đối tượng cũng dùng chỉ số để truy cập từng phần tử đối tượng, sau đó dùng toán tử (.) để truy cập hàm thành viên public:

  inventory[2].setUnits(30);

  cout << inventory[2].getUnits();

VD: MẢNG ĐỐI TƯỢNG

```
// Minh hoạ cách dùng một mảng đối tượng
#include <iostream>
#include <iomanip>
#include "InventoryItem.h"
using namespace std;

int main()
{
    const int NUM_ITEMS = 5;
    InventoryItem inventory[NUM_ITEMS] = {  InventoryItem("Hammer", 6.95, 12),
                                            InventoryItem("Wrench", 8.75, 20),
                                            InventoryItem("Pliers", 3.75, 10),
                                            InventoryItem("Ratchet", 7.95, 14),
                                            InventoryItem("Screwdriver", 2.50, 22) };

    cout << setw(14) <<"Inventory Item" << setw(8) << "Cost" << setw(8)
         << setw(16) << "Units on Hand\n";
    cout << "-----------------------------------\n";
    for (int i = 0; i < NUM_ITEMS; i++) {
        cout << setw(14) << inventory[i].getDescription(); cout << setw(8)
             << inventory[i].getCost();
        cout << setw(7) << inventory[i].getUnits() << endl;
    }
    return 0;
}
```

```
Inventory Item    Cost  Units on Hand
-----------------------------------
        Hammer    6.95     12
        Wrench    8.75     20
        Pliers    3.75     10
       Ratchet    7.95     14
    Screwdriver    2.5      22
```

VD: HOÀN THIỆN CHƯƠNG TRÌNH EMPLOYEE

- Sửa hàm main để tạo đối tượng và in ra theo cách dùng mảng

Name	ID Number	Department	Position
Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

ALGORITHM WORKBENCH

Circle Class

43. Viết khai báo lớp có tên **Circle** với một biến thành viên private có tên là **radius** và một hằng thành viên **pi** khởi tạo mặc định là 3.14159.
Viết ***setter*** và **gettter** để truy cập vào biến **radius** và một hàm **getArea** trả về diện tích hình tròn, tính bởi **pi** * radius * radius

44.Thêm một hàm tạo mặc định vào lớp Circle trong câu hỏi 43. Hàm này khởi tạo bán kính bằng 0.

45.Thêm một hàm tạo nạp chồng vào lớp Circle trong câu hỏi 44. Hàm tạo phải chấp nhận một đối số và gán giá trị của nó cho biến thành viên radius.

ALGORITHM WORKBENCH

Circle Class

46. Viết câu lệnh định nghĩa một mảng gồm năm đối tượng của lớp **Circle**. Cho hàm tạo mặc định thực thi cho mỗi phần tử của mảng.

47.Viết câu lệnh định nghĩa một mảng khác gồm năm đối tượng của lớp Circle và truyền các đối số sau đây cho hàm tạo của phần tử: 12, 7, 9, 14 và 8.

48.Viết vòng lặp for hiển thị bán kính và diện tích của các hình tròn được biểu diễn bởi mảng đã định nghĩa trong câu hỏi 47.

ALGORITHM WORKBENCH

Lớp **Personal Infor**

Thiết kế một lớp chứa thông tin cá nhân, gồm các thành viên sau:

- Tên kiểu chuỗi, địa chỉ kiểu chuỗi, tuổi kiểu nguyên, và số điện thoại kiểu chuỗi.
- Một setter tổng cập nhật cả 4 biến thành viên
- Một hàm in ra lần lượt cả 4 biến
- Một cấu tử mặc định gán tên, địa chỉ, số điện thoại là rỗng, tuổi là 0,
- 1 cấu tử 2 tham số cập nhật tên và tuổi, gán địa chỉ và số điện thoại rỗng
- Một cấu tử 4 tham số cập nhật cho cả 4 biến thành viên,

Hàm main minh hoạ lớp bằng cách tạo ra một mảng 3 đối tượng khởi tạo bằng 3 hàm tạo khác nhau. Sau đó in ra mọi thông tin của cả 3 đối tượng này.

ALGORITHM WORKBENCH

Lớp **Coin**

Viết lớp **Coin** có các thành viên sau:

- Biến **sideUp:** kiểu chuỗi, là "ngửa" hoặc "sấp" cho biết mặt của đồng xu
- Hàm tạo mặc định xác định ngẫu nhiên mặt của đồng xu ("ngửa" hoặc "sấp") và khởi tạo biến thành viên **sideUp** tương ứng.
- Hàm **toss** mô phỏng việc tung đồng xu. Khi **toss** được gọi, nó xác định ngẫu nhiên mặt của đồng xu ("ngửa" hoặc "sấp") và đặt biến thành viên **sideUp** cho phù hợp.
- Hàm **getSideUp** trả về giá trị của biến thành viên **sideUp**.

Minh hoạ lớp **Coin** bằng cách tạo một thể hiện của lớp và hiển thị mặt ban đầu của đồng xu. Sau đó, tung đồng xu 20 lần, mỗi lần lại hiển thị mặt của đồng xu. Đếm số lần ngửa, số lần sấp, và hiển thị kết quả khi vòng lặp kết thúc.

MỞ RỘNG

- Hàm tiện ích
- Hàm tiện ích kiểm tra dữ liệu
  - Ngoại lệ
  - Nhu cầu kiểm tra dữ liệu
  - Hàm kiểm tra dữ liệu
- Hàm tiện ích cập nhật biến dẫn xuất
  - Dữ liệu dẫn xuất: Hàm vs. Biến
  - Hàm setter biến dẫn xuất

HÀM TIỆN ÍCH

- Lập trình hướng cấu trúc gợi ý chia mã hàm lớn thành lời gọi đến vài hàm con cho dễ quản lý
- => OOP cũng cho phép chia hàm thành viên thành các hàm con.
  - các hàm con có thể thuộc hay không thuộc lớp
- Nếu hàm con chỉ để phục vụ các hàm trong lớp và không được gọi ở đâu khác, nó nên là hàm thuộc lớp mà để private. Hàm kiểu này gọi là hàm tiện ích (helper)

VD: HÀM TIỆN ÍCH CHO LỚP CAT

- Viết helper chase() mô phỏng bắt 1 con chuột, quy tắc như cũ, hàm trả về true nếu bắt được, false nếu không
- Dùng chase() để viết lại chaseMouse

```
private:
    string name;
    int age;
    float weight;
    bool chase() const{
        return rand() % 2;
    }
int Cat::chaseMouse(int n) const {
    srand(time(0));
    int count = 0;
    for (int i = 0; i<n; i++)
        if (chase()) count++;
    return count;
}
```

Cat
- name: string - age: int - weight: float
+ setInfor(string, int, float): void + getName() const: string + getAge() const: int + getWeight() const: float + meow(): void + chaseMouse(int): int

NHU CẦU KIỂM TRA DỮ LIỆU

Nếu nhập vào size âm??

```
class NumberArray
{
private:
    int size;
    double * a;
public:
    NumberArray(int size = 50)
    : size(size), a(new double [size]){}

    ~NumberArray() { delete [] a; }

    void setElement(double e, int i) { a[i] = e; }
    double getElement (int i) const { return a[i]; }
    double getMin() const;
    double getMax() const;
    double getAverage() const;
};
```

```
NumberArray(int size = 50)
: size(size), a(new double [size]){}

NumberArray(int s = 50){
    if (s < 0) {
        cout << "Loi\n";
        size = 50;
    }
    else size = s;
    a = new double [size];
}
NumberArray(int s = 50){
    if (s < 0) throw "Loi\n";
    size = s;
    a = new double [size];
}
```

Nếu cấp phát động với size âm??

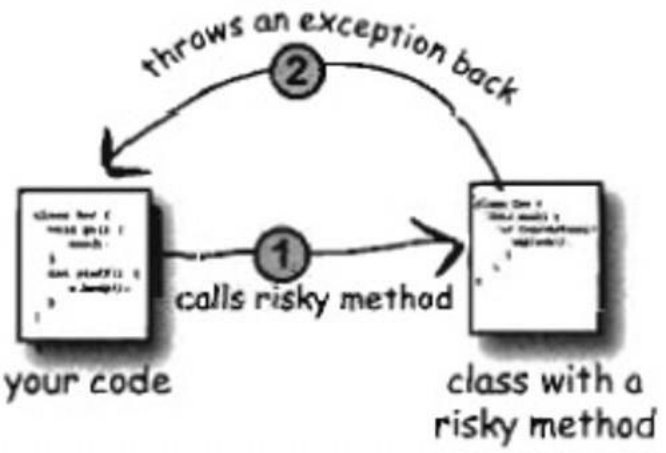
NGUY HIỂM: C++ thì báo lỗi, còn C thì chuyển số âm thành số dương rất lớn

Nhập kích thước âm thì tốt nhất không tạo mảng thay vì cố tạo ra một mảng mặc định

NGOẠI LỆ

- Xử lý lỗi kiểu PP: Phát hiện lỗi ở đâu, xử lý ngay ở đó
  - Logic xử lý lỗi bị trộn vào logic bài toán chính
  - Các đoạn mã xử lý lỗi đặt rải rác trong cả chương trình
  - Lỗi có lúc nên xử lý muộn hơn/ở cấp cao hơn
  - Có những lỗi thời gian chạy khó có thể trù tính (vd, tràn bộ nhớ)
- Xử lý lỗi kiểu OOP giải quyết các vấn đề trên:
  - Bao gói sự kiện lỗi trong các đối tượng có thể được "ném" từ nơi sinh ra lỗi đến nơi thuận tiện xử lý lỗi.
  - Thay "né lỗi" bằng "dọn lỗi", thay "dò lỗi" bằng "hứng lỗi"
  - Các đối tượng bao gói thông tin lỗi này gọi là các ngoại lệ

NÉM NGOẠI LỆ



- Kịch bản ném và bắt:
  - Một đoạn mã "ném" ra ngoại lệ, ngoại lệ này sẽ truyên lên các hàm cao hơn cho đến khi bị "bắt" bởi một đoạn mã nhận diện ra nó.
  - Nếu đến main vẫn chưa bị bắt thì sẽ làm kết thúc chương trình
- Ném: if <hoàn cảnh lỗi> throw <một ngoại lệ>
  - ngoại lệ có thể là một giá trị kiểu bất kỳ (số, xâu, đối tượng,…)
- Bắt: try { <mã nghi ngờ> }

  catch (<loại ngoại lệ muốn bắt> <biến để lưu ngoại lệ>)

  { <mã xử lý nếu bắt được ngoại lệ> }

```
NumberArray(int size = 50)
: size(size), a(new double [size]){}
```

Nếu cấp phát động với size âm??

NGUY HIỂM: C++ thì báo lỗi, còn C thì chuyển số âm thành số dương rất lớn

```
NumberArray(int s = 50){
    if (s < 0) {
        cout << "Loi\n";
        size = 50;
    }
    else size = s;
    a = new double [size];
}
```

Nhập kích thước âm thì tốt nhất không tạo mảng thay vì cố tạo ra một mảng mặc định

```
NumberArray(int s = 50){
    if (s < 0) throw "Loi\n";
    size = s;
    a = new double [size];
}
```

Cũng ok, nhưng nếu size phải khởi tạo kiểu danh sách thì sao? (VD có biến cần khởi tạo kiểu danh sách và nó lại truy xuất size => size không thể nằm ở thân hàm tạo)

Hơn nữa kiểm tra số âm có thể cần ở nhiều chỗ, nên biến mã này thành hàm để dễ tái sử dụng

```
inline int check (int s) const{
    if (s<=0) throw "Co cua mang phai lon hon 0";
    return s;
}
```

HÀM TIỆN ÍCH KIỂM TRA DỮ LIỆU

- Nếu dữ liệu không hợp lệ sẽ ném ngoại lệ, còn lại sẽ trả về dữ liệu này

```
inline int check (int s) const{
    if (s<=0) throw "Co cua mang phai lon hon 0";
    return s;
}
```

- Bắt ngoại lệ (thường đặt ở main):

```
try{
        đoạn_mã_cần_bắt_ngoại_lệ
}catch(const char * biến_lưu){    // ngoại lệ trên là 1 xâu
        đoạn_mã_xử_lý_ngoại_lệ
}
```

- Dùng được ở mọi hàm cập nhật dữ liệu (hàm setter, các hàm tạo)

VD:HÀM TIỆN ÍCH KIỂM TRA DỮ LIỆU

```
class NumberArray
{
private:
    int size;
    double * a;
    inline int check (int s) const{
        if (s<=0) throw "Co cua mang phai lon hon 0";
        return s;
    }
public:
    NumberArray(int size = 50): size(check(size)), a(new double [size]){}
    ~NumberArray() { delete [] a; }
    void setElement(double e, int i) { a[i] = e; }
    double getElement (int i) const { return a[i]; }
    double getMin() const;
    double getMax() const;
    double getAverage() const;
};
```

VD:HÀM TIỆN ÍCH KIỂM TRA DỮ LIỆU

```
int main(){
    try{
        int n;
        cout << "n: "; cin >> n;
        NumberArray arr(n);
        cout << arr.getSize() << endl;
    }catch (const char * e){
        cout << e;
    }
    cout << "Sau kiem tra ngoai le\n";
    return 0;

}
```

VD:HÀM TIỆN ÍCH KIỂM TRA DỮ LIỆU

Thiết kế lớp **Date** có các thành viên:

- Các biến thành viên month, day và year để biểu diễn một ngày trong năm
- Một hàm tạo nhận 3 tham số cập nhật cho 3 biến thành viên

Viết hàm main tạo 1 đối tượng Date.

Xác thực đầu vào dùng hàm tiện ích kiểm tra dữ liệu: Không chấp nhận day ngoài khoảng [1, 31], hay month ngoài khoảng [1, 12].

VD:HÀM TIỆN ÍCH KIỂM TRA DỮ LIỆU

```
class Date{
    int  day, month, year;
    int checkMonth(int m) const {
        if (m<1||m>12) throw "Thang sai\n";
        return m;
    }
    int checkDay(int d) const{
        if (d<1||d>31) throw "Ngay sai\n";
        return d;
    }
public:
    Date(int d, int m, int y)
    : day(checkDay(d)), month (checkMonth(m)), year(y){}
};

int main(){
    try{
        Date d(34, 15, 2022);
    }catch(const char * e){
        cout << e;
    }
    cout << "Chuong trinh ket thuc binh thuong \n";
    return 0;
}
```

```
Ngay sai
Chuong trinh ket thuc binh thuong
```

DỮ LIỆU DẪN XUẤT: HÀM VS. BIẾN?

- Là dữ liệu tính từ các dữ liệu khác (vd: diện tích chữ nhật)
- Nếu lưu như một biến **area**, thay đổi **length** hoặc **width** mà quên cập nhật **area**, thì **area** sẽ bị lạc hậu.
- Câu hỏi: Nên để dữ liệu dẫn xuất ở dạng biến hay hàm?
  - Nếu thường xuyên cần cập nhật, không nên lưu như biến, mà chỉ dùng hàm vd hàm **getArea()** trả về **width*length**
  - Nếu không hay cần cập nhật song lại thường xuyên cần truy vấn (vd ĐTB tích luỹ hàng kỳ) có thể lưu dữ liệu này như biến
    - Song nhớ đồng bộ hoá nó với các biến tạo thành (slide kế)

HÀM SETTER BIẾN DẪN XUẤT

- Nếu dùng biến dẫn xuất, cần có setter cho biến dẫn xuất:
  - Là setter không tham số (??)
  - Gọi ở mọi chỗ cập nhật các biến gốc
    - Gọi setArea không chỉ trong setWidth, setLength mà ở mọi hàm tạo có thay đổi 2 biến width, length
  - Luôn tự động gọi trong các hàm thành viên lớp, không bao giờ cần gọi từ ngoài lớp => là hàm tiện ích

VD: HÀM SETTER BIẾN DẪN XUẤT

Thiết kế lớp PayRoll có các thành viên:

- Lương theo giờ, số giờ làm việc trong tuần, tổng tiền lương trong tuần của một nhân viên.
- 1 hàm tạo 2 tham số với đối số mặc định
- Các setter, getter phù hợp.

Viết chương trình tạo một mảng bảy đối tượng Payroll, sau đó hiển thị số tiền lương mỗi nhân viên đã kiếm được.

Xác thực đầu vào: Không chấp nhận số giờ làm việc lớn hơn 60.

VD: SETTER BIẾN DẪN XUẤT

```
class Payroll{
    double rate;    // lương theo giờ
    int hour;       // số giờ làm việc trong tuần
    double payment; // lương tuần payment = rate * hour
    int checkHour(int h) const{
        if (h>60) throw "Loi: so gio khong duoc > 60\n";
        return h;
    }
    void setPayment(){ payment = hour * rate; }
public:
    Payroll(double rate = 2.1, int hour = 40)
    : rate(rate), hour(checkHour(hour))
    { setPayment(); }
    void setRate(double r) {
        rate = r;
        setPayment();
    }
    void setHour(int h){
        hour = checkHour(h);
        setPayment();
    }
```

VD: SETTER BIẾN DẪN XUẤT

```
    double getRate() const { return rate; }
    int getHour() const { return hour; }
    double getPayment() const { return payment; }
};

int main(){
    const int n = 7;
    // điền 4 phần tử trong {} song thực sự gọi 7 hàm tạo
    Payroll a[n] = {Payroll(6, 48), Payroll(7, 40), 6, 2};

    for (int i = 0; i<n; i++)
        cout << a[i].getRate() << " " << a[i].getHour()
             << " " << a[i].getPayment() << endl;
    return 0;
}
```

```
6 48 288
7 40 280
6 40 240
2 40 80
2.1 40 84
2.1 40 84
2.1 40 84
```

TỔNG KẾT

- Cú pháp và các từ khoá khai báo lớp cơ bản
- Định nghĩa hàm ở trong và bên ngoài khai báo lớp
- Định nghĩa các loại hàm quan trọng: setter, getter, constructor, destructor, helper
- Mở rộng: hàm tiện ích kiểm tra dữ liệu, cập nhật biến dẫn xuất

CÂU HỎI REVIEW

Keep calm and carry on