



Chương III. LỚP VÀ ĐỐI TƯỢNG

1. Object (đối tượng)	28
2. Class (lớp)	28
3. Khai báo và sử dụng lớp	29
4. Khai báo và sử dụng đối tượng	31
5. Chuẩn hóa mã nguồn	33
6. Các loại phương thức của lớp	33
6.1 Phương thức khởi tạo (constructor)	33
6.2 Phương thức hủy đối tượng (destructor)	37
6.3 Phương thức get/set	39
7. Thành phần tĩnh (static)	40
7.1 Thành phần dữ liệu tĩnh	40
7.2 Thành viên dữ liệu tĩnh	40
7.3 Phương thức thành viên tĩnh (Hàm thành phần tĩnh)	41
8. Thành phần hằng (const)	43
8.1 Thành viên dữ liệu const	43
8.2 Phương thức thành viên const	43
8.3 Đối tượng hằng (const object)	44
9. Con trỏ this	44
10. Hàm bạn, lớp bạn	45
10.1 Friend	45
10.2 Hàm bạn	46
10.3 Lớp bạn	47
11. Đối tượng là thành phần của lớp	48
12. Đối tượng là thành phần của mảng	50
13. Các nguyên tắc xây dựng lớp	51

1. Object (đối tượng)



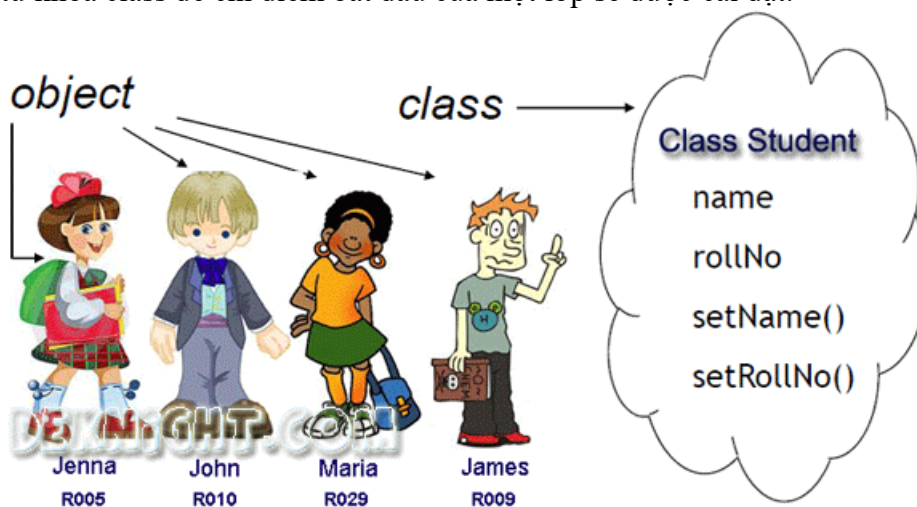
Object = Data + Methods

 LightBulb	<ul style="list-style-type: none">- state/attributes on (true or false)- behavior switch on switch off check if on	 BankAccount	<ul style="list-style-type: none">- state/attributes balance- behavior Deposit (tiền gửi) Withdraw (tiền rút) check balance
---	---	---	--

- Mỗi đối tượng là thể hiện của lớp
- Mỗi thể hiện sẽ có trạng thái khác nhau
Ví dụ: hai tài khoản khác nhau sẽ có balance khác nhau
- Đối tượng là công cụ hỗ trợ sự đóng gói dữ liệu

2. Class (lớp)

- 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.
- Là đại diện cho một tập các đối tượng có cùng thuộc tính và hành vi
- Lớp là kiểu dữ liệu trừu tượng (ADT).
- 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.



3. Khai báo và sử dụng lớp

<pre>class class_name { access_specifier: member1; access_specifier: member2; ... };</pre>	<ul style="list-style-type: none">- access_specifier: quyền truy cập, chỉ định mức độ cho phép truy cập (tính bảo mật)- Các giới hạn truy cập:<ul style="list-style-type: none">+ public: mọi nơi nếu đối tượng tồn tại (trong và ngoài lớp); thành viên được truy xuất tùy ý (tài sản công cộng)+ private: trong phạm vi của lớp; riêng tư (chỉ chủ nhân dùng)+ protected: trong phạm vi của lớp và các lớp con thừa kế; giữa public & private (1 số người quen với chủ nhân thì được dùng chung tài sản này)- 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.- Hàm thành phần có quyền truy cập đến các thành phần private của đối tượng gọi nó
--	---

*** Lưu ý: Sự khác biệt giữa từ khóa **struct** và **class**:

Trong một lớp có thể không có hoặc 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.

Nếu khai báo một Lớp sử dụng từ khóa **struct**, những thành phần được khai báo trước nhãn truy cập đầu tiên sẽ được mặc định là **public**, nếu sử dụng từ khóa **class**, những thành phần đó sẽ mặc định là **private**:

```
class HocSinh {
    int mssv;
    string hoTen; // MSSV và HoTen được xem như private
    double diemToan; double diemVan;
    // ...
    // những thành phần còn lại như ví dụ ở trên
};
struct HocSinh {
    int mssv;
    string hoTen; // MSSV và HoTen được xem như public
    double diemToan; double diemVan;
    // ...
    // những thành phần còn lại như ví dụ ở trên
};
```

Vì vậy khi khai báo một Lớp mà không chỉ định bất kì phạm vi truy xuất nào, thì tất cả các thuộc tính và phương thức sẽ mặc định là **public** nếu sử dụng **struct**, và là **private** nếu sử dụng **class**. Lưu ý rằng đây cũng là điểm khác biệt duy nhất giữa **struct** và **class**.

- Data members (variable): type name; (Khai báo trong file.h)

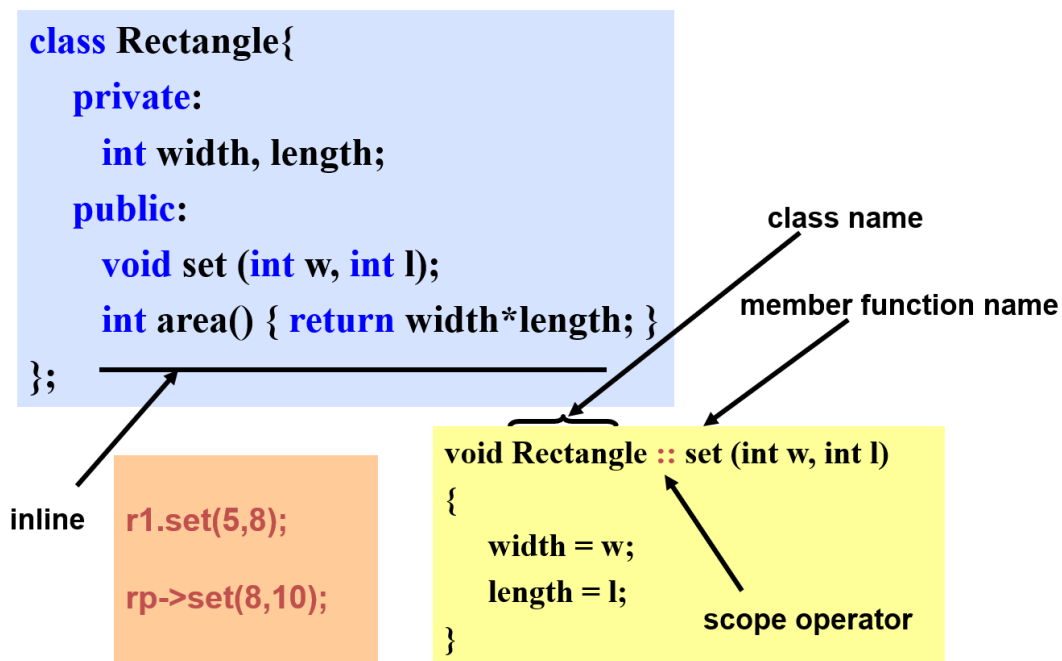
Ví dụ 1: `double balance;` // Tương tự như khai báo biến
+ member functions (method)

- Khai báo: return_type func(type arg1, type arg2,...); (Khai báo trong file.h) (Tương tự như khai báo nguyên mẫu hàm)

- Định nghĩa: (trong file.cpp) (Tương tự như định nghĩa hàm)

```
return_type class_name::func(type arg1, type arg2,...)
{
    //body of function
}
```

Ví dụ 2:



Ví dụ 3:

<p>* Mô tả lớp CRectangle.</p> <pre> 1 //Rectangle.h 2 #pragma once 3 4 class CRectangle 5 { 6 private: 7 int width, height; 8 public: 9 void setWidth(int _width); 10 int getWidth() const; 11 void setHeight(int _height); 12 int getHeight() const; 13 int area(); 14 }; 15 </pre>	<p>* 2 data members</p> <p>private:</p> <p>int width;</p> <p>private:</p> <p>int height;</p> <p>* 5 member function: line 9 → 13</p> <p>→ Total: 7 members</p> <p>* Từ khóa const: Không thay đổi giá trị của các data members.</p>
<pre> 1 //Rectangle.cpp 2 #include "Rectangle.h" 3 int CRectangle::getHeight() const 4 { 5 return height; 6 } 7 int CRectangle::getWidth() const 8 { 9 return width; 10 } 11 void CRectangle::setHeight(int _height) 12 { 13 height = _height; 14 } 15 void CRectangle::setWidth(int _width) 16 { 17 width = _width; 18 } 19 int CRectangle::area() 20 { 21 return width*height; 22 } 23 </pre> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>Phương thức getWidth() nằm trong phạm vi lớp</p> </div> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>Thành viên dữ liệu "height" được phép truy cập trực tiếp trong phương thức thành viên setHeight()</p> </div> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px; width: 45%; margin-left: auto;"> <p>Không bắt buộc dùng con trỏ this do compiler phân biệt được Member này của lớp nào</p> </div>	

Ví dụ 4: Xây dựng và sử dụng lớp **Student**

+ Data members: **mssv, name, averMark**

+ Member functions: **print(), input()**

Ví dụ 5: Xây dựng và sử dụng lớp **Fraction**

+ Data members: **numerator, denominator**

+ Member functions: **print(), input(), add()**

```

int Trung (point pt){
    return (x==pt.x && y==pt.y);
}
int Trung (point *pt){
    return (x==pt->x && y==pt->y);
}
int Trung (point &pt) {
    return (x==pt.x && y==pt.y);
}

```

- Hàm thành phần có quyền truy cập đến tất cả các thành phần **private** của các đối tượng, tham chiếu đối tượng hay con trỏ đối tượng có cùng kiểu lớp khi được dùng là tham số hình thức của nó.

4. Khai báo và sử dụng đối tượng

- Cú pháp: **<tên lớp> <tên đối tượng>;**

class_name object1, object 2, ...;

class_name *object3;

Ví dụ 1.1: Student s1, s2;
 CRectangle rect1, *rect2;

- Truy xuất thành viên:

<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>;

object1.member1;

object1.member2;

object3->member1;

Ví dụ 1.2: s1.input();
 rect2->setWidth(2);

Ví dụ 2:

```

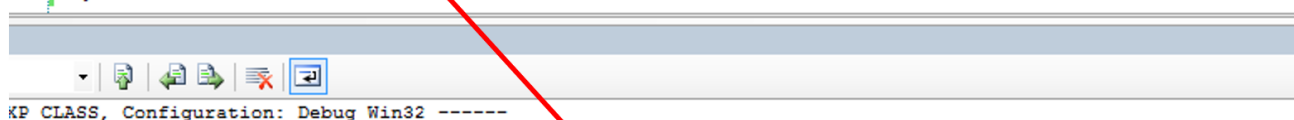
1 //main.cpp
2 #include <iostream>
3 #include "Rectangle.h"
4 using namespace std;
5
6 int main()
7 {
8     CRectangle rect;
9     rect.height = 2;
10    rect.setHeight(3);
11    rect.setWidth(4);
12    cout<<"Diện tích: "<< rect.area()<<endl;
13    return 0;
14 }

```

Tạo đối tượng rect

Phải thông qua phương thức setHeight để gán giá trị cho height

Truy xuất hàm thành viên



main.cpp(9) : error C2248: 'CRectangle::height' : cannot access private member declared in class 'CRectangle'

```

1 //main.cpp
2 #include <iostream>
3 #include "Rectangle.h"
4 using namespace std;
5 int main() {
6     CRectangle rect1, *rect2;
7     //rect1.height = 2;
8     rect1.setHeight(3);
9     rect1.setWidth(4);
10    cout<<"Height1: "<<rect1.getHeight()<<endl;
11    cout<<"Width1: "<<rect1.getWidth()<<endl;
12    cout<<"S1: "<< rect1.area()<<endl;
13    cout<<endl;
14    rect2 = new CRectangle();
15    rect2->setHeight(5);
16    rect2->setWidth(4);
17    cout<<"Height2: "<<rect2.getHeight()<<endl;
18    cout<<"Width2: "<<rect2->getWidth()<<endl;
19    cout<<"S2: "<< rect2->area()<<endl;
20    return 0;
21 }

```

Đối tượng là 1 con trỏ

Cấp bộ nhớ động cho con trỏ rect2

Truy xuất hàm thành viên

from: Build

:\user\desktop\exp_class\main.cpp(19) : error C2228: left of '.getHeight' must have class/struct/union type is 'CRectangle *' did you intend to use '->' instead?

<pre> #include <iostream> using namespace std; class CRectangle { int width, height; public: void setWidth(int _width); void setHeight(int _height); int getWidth() const; int getHeight()const; int area(); }; void CRectangle::setWidth(int _width) { width = _width; } void CRectangle::setHeight(int _height) { height = _height; } int CRectangle::getWidth() const { return width; } int CRectangle::getHeight() const { return height; } int CRectangle::area() { return width*height; } int main() { CRectangle rect1, * rect2; rect1.setHeight(3); rect1.setWidth(4); cout << "Height1: " << rect1.getHeight() << endl; cout << "Width1: " << rect1.getWidth() << endl; cout << "S1: " << rect1.area() << endl; cout << endl; rect2 = new CRectangle(); rect2->setHeight(5); rect2->setWidth(4); </pre>	<p>Kết quả:</p> <p>Height1: 3 Width1: 4 S1: 12</p> <p>Height2: 5 Width2: 4 S2: 20</p>
--	--

```

cout << "Height2: " << rect2->getHeight() << endl;
cout << "Width2: " << rect2->getWidth() << endl;
cout << "S2: " << rect2->area() << endl;
delete rect2;
return 0;
}

```

5. Chuẩn hóa mã nguồn

- Khai báo class trong header file

“Rectangle.h” (*interface/ Giao diện – Bề mặt*)

- Định nghĩa tất cả các phương thức trong file

“Rectangle.cpp” (*implementation/ Phương thức – Cài đặt – Ẩn giấu*)

- Để tránh include nhiều lần khai báo class (Rectangle.h) sử dụng:

```

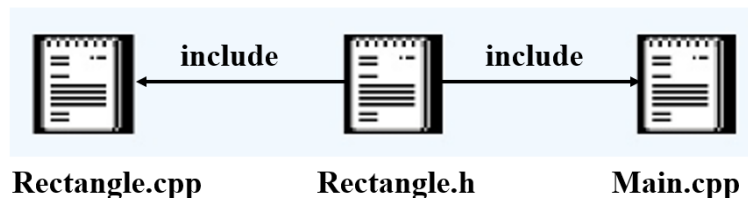
#ifndef _RECTANGLE_H
#define _RECTANGLE_H
//Class declaration
....

```

#endif

Hoặc: **#pragma once**

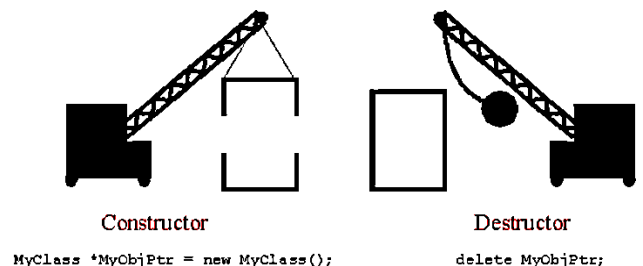
- Sử dụng class, tạo đối tượng trong file: **“Main.cpp”** (*client*)



6. Các loại phương thức của lớp

- Mỗi đối tượng thường có 4 phương thức cơ bản:

- + Phương thức khởi tạo đối tượng: **constructor**
- + Phương thức hủy đối tượng: **destructor**
- + Phương thức truy xuất dữ liệu: **get**
- + Phương thức cập nhật dữ liệu: **set**



6.1 Phương thức khởi tạo (constructor)

- Giả sử ta cần gán các thuộc tính của đối tượng bằng các giá trị cụ thể nào đó. Chẳng hạn như ví dụ 2/ phần 4, ta sử dụng các hàm setHeight & setWidth để gán giá trị cho các thuộc tính của 2 đối tượng rect1 & *rect2. Tuy nhiên, nếu có thêm n đối tượng khác nữa được khai báo, thì số lượng hàm setWidth & setHeight sẽ tăng lên 2n lời gọi hàm, khiến hàm main thêm rối. Do vậy, ta cần sử dụng một hàm để khởi tạo 1 lần các giá trị cho các thuộc tính của từng đối tượng, đó là constructor.

- 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 phải có thuộc tính public**

- Chức năng:

- + Khởi tạo các giá trị thành viên dữ liệu của đối tượng
- + Xin cấp phát bộ nhớ cho thành viên dữ liệu động

- Là hàm thành viên của lớp

- Nó **được gọi tự động** mỗi khi đối tượng được khai báo

- Chỉ chạy 1 lần duy nhất.

- Khai báo

class_name(type arg1, type arg2,...);

- Định nghĩa

Ta định nghĩa một phương thức bên trong thân lớp tương tự như khi định nghĩa một hàm bình thường, còn khi muốn định nghĩa một phương thức bên ngoài lớp, ta phải sử dụng **toán tử phạm vi** (scope operator – dấu ::) để chương trình biết ta đang định nghĩa phương thức của lớp nào. Lúc này phần thân phương thức được xem như đang nằm trong phạm vi của lớp đó.

```
class_name::class_name(type arg1, type arg2,...)
{
    //Thân hàm
}
```

Ví dụ 1:

```
1 //Rectangle.h
2 #pragma once
3
4 class CRectangle
5 {
6 private:
7     int width, height;
8 public:
9     CRectangle(int, int);
10    void setWidth(int _width);
11    int getWidth() const;
12    void setHeight(int _height);
13    int getHeight() const;
14    int area();
15 };
16
```

Khai báo phương thức khởi tạo hai tham số

```
1 //Rectangle.cpp
2 #include "Rectangle.h"
3 CRectangle::CRectangle(int w, int h)
4 {
5     width = w;
6     height = h;
7 }
8 int CRectangle::getHeight() const
9 {
10    return height;
11 }
12 int CRectangle::getWidth() const
13 {
14    return width;
15 }
16 void CRectangle::setHeight(int _height)
17 {
18    height = _height;
19 }
```

Định nghĩa phương thức khởi tạo hai tham số

- Một số đặc điểm của phương thức khởi tạo:

- + Có cùng tên với tên lớp
- + Không có giá trị trả về
- + Có thể có nhiều phương thức khởi tạo trong cùng lớp (chồng hàm)
- + Có thể khai báo với tham số có giá trị ngầm định

- Một số phương thức khởi tạo

+ Default Constructor

- . Không có tham số
- . Chương trình *tự động phát sinh* nếu trong lớp không xây dựng phương thức khởi tạo nào

+ Parameterized Constructor

- . Có một hoặc nhiều tham số
- . Đối số được dùng để khởi tạo đối tượng

Ví dụ 2:

```
1 //main.cpp
2 #include <iostream>
3 #include "Rectangle.h"
4 using namespace std;
5 int main() {
6     CRectangle rect1, *rect2;
7     //rect1.height = 2;
8     rect1.setHeight(3);
9     rect1.setWidth(4);
10    cout<<"Height1: "<<rect1.getHeight()<<endl;
11    cout<<"Width1: "<<rect1.getWidth()<<endl;
12    cout<<"S1: "<< rect1.area()<<endl;
13    cout<<endl;
14
15    rect2 = new CRectangle();
16    rect2->setHeight(5);
17    rect2->setWidth(4);
18    cout<<"Height2: "<<rect2->getHeight()<<endl;
19    cout<<"Width2: "<<rect2->getWidth()<<endl;
20 }
```

```
1 //Rectangle.h
2 #pragma once
3
4 class CRectangle
5 {
6 private:
7     int width, height;
8 public:
9     CRectangle(int, int);
10    void setWidth(int _width);
11    int getWidth() const;
12    void setHeight(int _height);
13    int getHeight() const;
14    int area();
15 };
16
```

Khai báo phương thức khởi tạo hai tham số

lass\main.cpp(6) : error C2512: 'CRectangle' : no appropriate default constructor available
lass\main.cpp(15) : error C2512: 'CRectangle' : no appropriate default constructor available


```

1 //main.cpp
2 #include <iostream>
3 #include "Rectangle.h"
4 using namespace std;
5 int main(){
6     CRectangle rect1(3,4);
7     //rect1.height = 2;
8     cout<<"Height1: "<<rect1.getHeight()<<endl;
9     cout<<"Width1: "<<rect1.getWidth()<<endl;
10    cout<<"S1: "<< rect1.area()<<endl;
11    cout<<endl;
12
13    CRectangle *rect2 = new CRectangle(5,4);
14    cout<<"Height2: "<<rect2->getHeight()<<endl;
15    cout<<"Width2: "<<rect2->getWidth()<<endl;
16    cout<<"S2: "<< rect2->area()<<endl;
17    return 0;
18 }

```

Phương thức khởi tạo hai tham số được gọi

```

1 //Rectangle.h
2 #pragma once
3
4 class CRectangle
5 {
6 private:
7     int width, height;
8 public:
9     CRectangle();
10    CRectangle(int, int);
11    void setWidth(int _width);
12    int getWidth() const;
13    void setHeight(int _height);
14    int getHeight() const;
15    int area();
16 };

```

```

1 //Rectangle.cpp
2 #include "Rectangle.h"
3 CRectangle::CRectangle()
4 {
5     width = 1;
6     height = 1;
7 }
8 CRectangle::CRectangle(int w, int h)
9 {
10    width = w;
11    height = h;
12 }
13 int CRectangle::getHeight() const
14 {
15    return height;
16 }
17 int CRectangle::getWidth() const
18 {
19    return width;
20 }

```

```

4 using namespace std;
5 int main(){
6     CRectangle rect1(3,4);
7     //rect1.height = 2;
8     cout<<"Height1: "<<rect1.getHeight()<<endl;
9     cout<<"Width1: "<<rect1.getWidth()<<endl;
10    cout<<"S1: "<< rect1.area()<<endl;
11    cout<<endl;
12    CRectangle *rect2 = new CRectangle(5,4);
13    cout<<"Height2: "<<rect2->getHeight()<<endl;
14    cout<<"Width2: "<<rect2->getWidth()<<endl;
15    cout<<"S2: "<< rect2->area()<<endl;
16    cout<<endl;
17    CRectangle rect3;
18    cout<<"Height3: "<<rect3.getHeight()<<endl;
19    cout<<"Width3: "<<rect3.getWidth()<<endl;
20    cout<<"S3: "<< rect3.area()<<endl;
21    return 0;
22 }

```

Hai phương thức khởi tạo đối tượng

+ Copy Constructor (Trường hợp đặc biệt của Parameterized Constructor) (Khởi tạo sao chép)

. Có **một tham số thuộc kiểu class đang khai báo**.

. Sao chép thành viên dữ liệu của một đối tượng cho

đối tượng khác

. 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 tham chiếu đến đối tượng thuộc chính lớp này**.

. 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”

. Khai báo

class name(const class name &arg)

. Định nghĩa

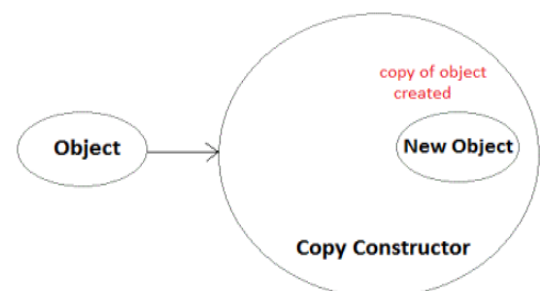
```

class_name::class_name(const class_name &arg)
{
    //thân hàm
    ....
}

```

. Hàm khởi tạo sao chép trong C++ thường có từ khóa const vì nó cho phép đảm bảo rằng đối tượng được sao chép không bị thay đổi.

. Khi bạn tạo một hàm khởi tạo sao chép, bạn thường truyền vào một tham chiếu đến đối tượng bạn muốn sao chép. Nếu bạn không sử dụng từ khóa const, thì bạn có thể thay đổi đối tượng đó trong hàm khởi tạo sao chép, điều này có thể dẫn đến các lỗi và hành vi không mong muốn.



* Trước hết chúng ta sẽ xem qua ví dụ về việc định nghĩa một phương thức thiết lập sao chép trong lớp

HocSinh:

```
HocSinh(const HocSinh& temp) {  
    cout << "Copy constructor of HocSinh has been called" << endl; mssv = temp.mssv;  
    hoTen = temp.hoTen;  
    diemToan = temp.diemToan; diemVan = temp.diemVan; diemTB = temp.diemTB;  
}
```

Trong chương trình, ta gọi thực hiện phương thức sao chép bằng cách khai báo:

```
HocSinh hs2(hs);
```

Hoặc:

```
HocSinh hs2 = hs;
```

Lúc này chương trình sẽ tự động thực hiện dòng lệnh sau:

```
const HocSinh &temp = hs;
```

Chương trình tới đây sẽ **không sao chép dữ liệu** của hs vào temp mà chỉ xem temp như là một **cái tên khác** của hs, những thao tác lúc này được thực hiện bên trong thân phương thức ở trên chính là gán các giá trị của hs cho hs2.

Tới đây có thể nhận thấy rằng nếu temp không được khai báo là tham chiếu mà chỉ là một biến bình thường thì chương trình sẽ ngầm thực hiện dòng lệnh:

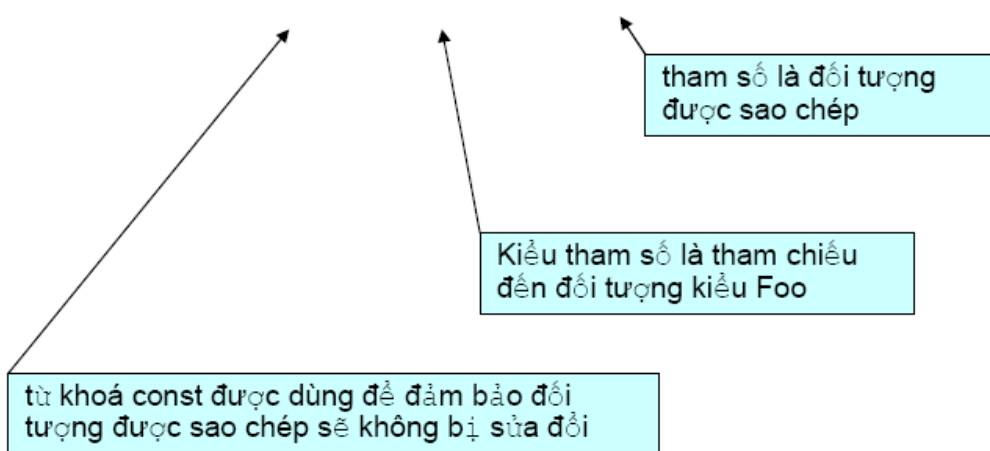
```
const HocSinh temp = hs;
```

Lúc này, trong quá trình thực hiện phương thức thiết lập sao chép để sao chép hs vào hs2, chương trình lại phải gọi thêm một phương thức thiết lập sao chép khác để sao chép hs vào temp, và cứ như vậy tạo thành một **vòng lặp vô hạn**.

Trong phương thức trên, ta khai báo tham chiếu temp là hằng để đảm bảo đối số truyền vào **không thể bị sửa đổi** một cách vô ý, cũng như đảm bảo rằng có thể truyền vào phương thức một **đối số hằng**.

Ví dụ 3:

```
Foo(const Foo& existingFoo);
```



Ví dụ 4:

```

1 //Rectangle.cpp
2 #include "Rectangle.h"
3 CRectangle::CRectangle()
4 {
5     width = 1;
6     height = 1;
7 }
8 CRectangle::CRectangle(int w, int h)
9 {
10    width = w;
11    height = h;
12 }
13 CRectangle::CRectangle(const CRectangle &r)
14 {
15    width = r.width;
16    height = r.height;
17 }

```

Sao chép dữ liệu thành viên của đối tượng r cho đối tượng hiện thời

+ Được sử dụng để copy **một phần/ toàn bộ** dữ liệu thành viên của một đối tượng cho một đối tượng khác có cùng kiểu.

+ Copy constructor được gọi trong các tình huống sau:

```

CRectangle rect3 = rect1;
CRectangle rect3;
rect3 = rect1;
CRectangle rect3(rect1);
CRectangle arr[2] = {rect};

```

Trong quá trình lập trình, chúng ta hoàn toàn có thể gán một đối tượng cho một đối tượng khác thuộc cùng lớp, khi đó 2 đối tượng sẽ hoàn toàn giống nhau về giá trị (tất cả các byte).

. Một số lý do khác cần sử dụng hàm khởi tạo sao chép:

Quản lý bộ nhớ: Khi bạn có một lớp mà các đối tượng của nó quản lý bộ nhớ (ví dụ: thông qua con trỏ), hàm khởi tạo sao chép cho phép bạn đảm bảo rằng mỗi đối tượng có bản sao riêng của dữ liệu mà nó quản lý. Điều này giúp tránh các vấn đề như việc giải phóng bộ nhớ hai lần.

Tạo bản sao của đối tượng: Trong một số trường hợp, bạn có thể muốn tạo một bản sao hoàn chỉnh của một đối tượng, ví dụ: để thực hiện một thao tác mà không làm thay đổi đối tượng gốc.

Truyền đối tượng vào hàm: Khi bạn truyền một đối tượng vào một hàm theo giá trị (thay vì theo tham chiếu), hàm khởi tạo sao chép sẽ được gọi để tạo một bản sao của đối tượng đó.

Trả về đối tượng từ hàm: Khi bạn trả về một đối tượng từ một hàm, hàm khởi tạo sao chép sẽ được gọi để tạo một bản sao của đối tượng đó.

6.2 Phương thức hủy đối tượng (destructor)

- Chức năng:

- + Hủy bỏ đối tượng khi không sử dụng nó nữa
- + Giải phóng bộ nhớ đã cấp phát động cho các thành viên dữ liệu (xóa các bộ nhớ do con trỏ tạo ra)
- + Đóng các file, hủy các file tạm
- + Đóng các kết nối mạng, kết nối cơ sở dữ liệu,...

- Là hàm thành viên của lớp

- Nó **được gọi tự động** mỗi khi đối tượng bị hủy bỏ

- Chỉ chạy 1 lần duy nhất.

- Khai báo

~class_name()

- Định nghĩa

```

class_name::~~class_name()
{
    //Than ham
}

```

Ví dụ:

```

1 #pragma once
2
3 class Trainee
4 {
5 private:
6     int id;
7     char *name;
8 public:
9     int getID() const;
10    char* getName() const;
11    void setID(int _id);
12    void setName(char *n);
13    Trainee(void);
14    Trainee(int _id, char *n);
15    ~Trainee(void);
16 };

```

```

2 #include <string.h>
3 Trainee::Trainee(void)
4 {
5     id = 0;
6     name = NULL;
7 }
8 Trainee::Trainee(int _id, char *n)
9 {
10    id = _id;
11    name = new char[strlen(n)+1];
12    strcpy(name, n);
13 }
14 Trainee::~~Trainee(void)
15 {
16    if(name != NULL)
17        delete []name;
18 }
19 int Trainee::getID() const
20 {

```

“name” được cấp phát bộ nhớ động ở hàm tạo và giải phóng vùng nhớ ở hàm hủy

- Một số đặc điểm của phương thức hủy:

- + Có cùng tên với tên lớp và bắt đầu bằng dấu ~
- + Không có giá trị trả về, không có tham số
- + Destructor phải có thuộc tính **public**
- + **Mỗi lớp chỉ có duy nhất một phương thức hủy**
- + Chương trình tự động phát sinh phương thức hủy nếu nó không được định nghĩa tường minh

- Phương thức phá hủy và phương thức thiết lập sao chép

- + Khi thiết kế các lớp đối tượng, có một quy luật là nếu một lớp cần phải tự định nghĩa phương thức phá hủy, thì lớp đó cũng cần tự định nghĩa một phương thức thiết lập sao chép của riêng mình.
- + Trong ví dụ về lớp **LopHoc** ở trên, nếu không làm gì thêm, chương trình sẽ tự định nghĩa cho ta một phương thức thiết lập sao chép như sau:

```

1. LopHoc(const LopHoc& temp) {
2.     this->arr = temp.arr; //sau bước này this->arr và temp.arr
3.     //cùng nắm giữ một vùng nhớ
4.     this->size = temp.size;
5. }

```

+ Tại dòng số 2, địa chỉ mà biến-con-trỏ-arr-thuộc-đối-tượng-temp đang nắm giữ được sao chép vào biến con trỏ arr của đối tượng đang thực hiện lời gọi phương thức. Lúc này hai con trỏ có giá trị bằng nhau, tức là chúng đang **cùng nắm giữ một vùng nhớ**. Khi thực hiện đoạn chương trình sau đây:

```

1. LopHoc item1(5);
2. {
3.     LopHoc item2 = item1; // Copy constructor do chương trình
4.                           // tự định nghĩa được gọi.
5. } // Destructor được gọi trên item2.
6. item1.Xuat(); // lỗi! item1 bây giờ bị mất vùng nhớ.

```

Trong phạm vi từ dòng số 2 đến dòng số 5, item2 được khởi tạo bằng copy constructor do chương trình tự định nghĩa, sao chép dữ liệu từ item1. Khi item2 ra khỏi phạm vi, destructor của lớp **LopHoc** được gọi trên item2 thu hồi vùng nhớ được cấp phát cho biến này nhưng cũng “vô tình” thu hồi vùng nhớ được cấp phát cho item1 vì thành phần arr trong hai đối tượng này đang cùng sở hữu một vùng nhớ.

+ Để tránh lỗi trên, ta cần phải định nghĩa cho lớp **LopHoc** một phương thức thiết lập sao chép như dưới đây:

```

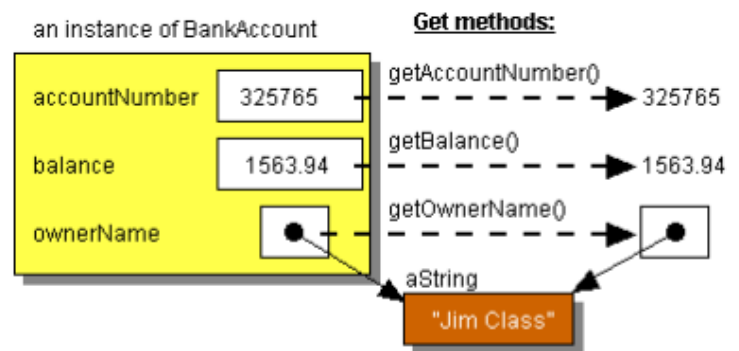
LopHoc(const LopHoc& temp) {
    size = temp.size;
    arr = new HocSinh[size];
    for (int i = 0; i < size; ++i) {
        arr[i] = temp.arr[i];
    }
}

```

- + Lúc này, thành phần arr trong 2 biến item1 và item2 sẽ giữ địa chỉ của 2 vùng nhớ khác nhau, các thao tác trên đối tượng này sẽ không ảnh hưởng tới đối tượng kia.
- + Ví dụ trên cho thấy không phải lúc nào các phương thức do chương trình tự định nghĩa cũng hoạt động như ý chúng ta mong muốn, nên ta cần phải chú ý khi nào thì nên tự định nghĩa các phương thức thiết lập của riêng mình.
- + Trong thực tế, việc quản lý bộ nhớ cho các đối tượng được cấp phát động thường gây khó khăn và dễ phát sinh ra lỗi, vì thế ngôn ngữ C++ có hỗ trợ cho ta các thư viện và lớp đối tượng được cài đặt sẵn như **vector** (có công dụng như một mảng động), **shared_ptr** (con trỏ tự thu hồi bộ nhớ), ... để việc lập trình trở nên dễ dàng hơn.

6.3 Phương thức get/set

- Dữ liệu thành viên: **private**
- Phương thức thành viên: **public**
- **Cần định nghĩa 2 phương thức get/set**
- 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.

* *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 được sử dụng để lấy giá trị của một thuộc tính **private***

• *Phương thức cập nhật dùng để thay đổi giá trị của một thuộc tính **private***

* 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

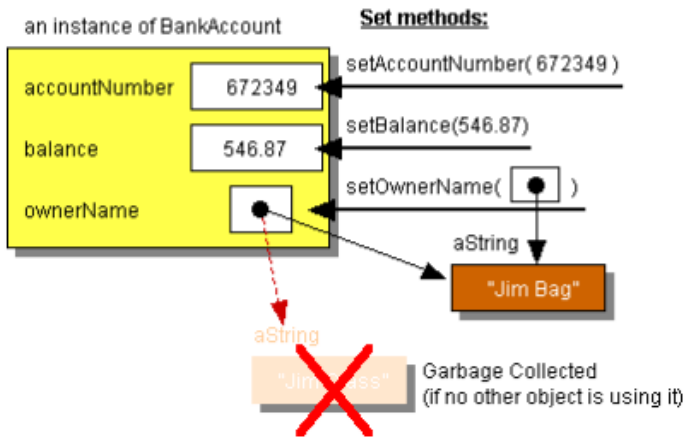
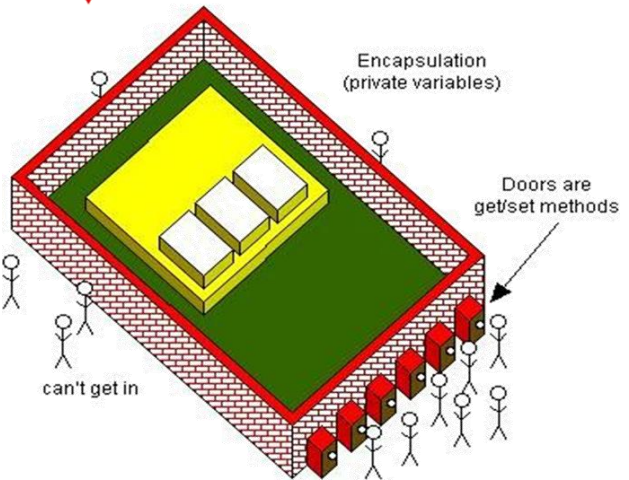
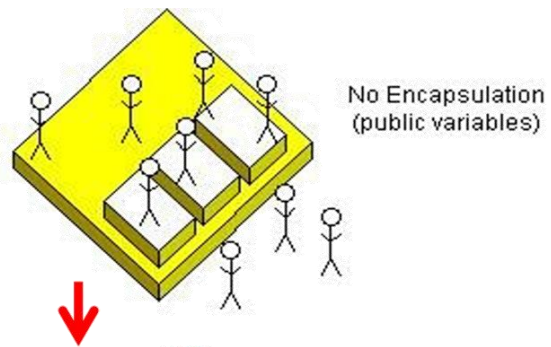
- Thường để **thay đổi trạng thái của đối tượng** bằng cách sửa đổi một hoặc nhiều thành viên dữ liệu của đối tượng đó.

- 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);`

<pre><fieldType> get<FieldName>() { return <fieldName>; }</pre>	<pre>void set<FieldName>(<fieldType> <paramName>) { <fieldName> = <paramName>; }</pre>
---	--



7. Thành phần tĩnh (static)

7.1 Thành phần dữ liệu tĩnh

- 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 thuộc tính trong lớp được khai báo bằng từ khóa **static** được gọi là thành phần **dữ liệu tĩnh**. Các thuộc tính này được cấp phát một vùng nhớ cố định, tồn tại ngay cả khi lớp chưa có một đối tượng nào cả. Dữ liệu tĩnh là thành phần chung cho cả lớp, không của riêng từng đối tượng.

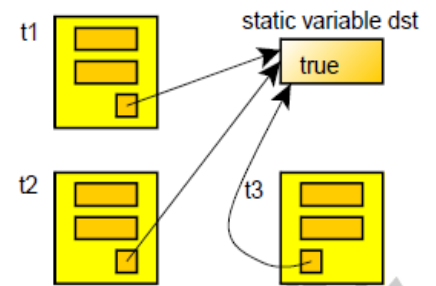
- Static trong c++ là dữ liệu của lớp không phải là dữ liệu của đối tượng. Static trong c++ tồn tại như 1 biến toàn cục. Hay nói cách khác dữ liệu static xuất hiện trước lúc bạn khởi tạo đối tượng của lớp, và nó chỉ tồn tại duy nhất.

- Các thành viên **static** có thể là **public**, **private** hoặc **protected**.

- Giá trị của thành viên dữ liệu được chia sẻ cho tất cả các đối tượng của lớp

- Dữ liệu tĩnh được cấp phát bộ nhớ 1 lần duy nhất

- Phải được khởi tạo bên ngoài khai báo lớp, ngoài tất cả các hàm



7.2 Thành viên dữ liệu tĩnh

- Khai báo:

```
static datatype var;
```

Ví dụ: static int count;

- Khởi tạo:

```
datatype class_name::var = value;
```

- Truy xuất:

+ Theo đối tượng:

```
CRectangle a;
```

```
a.count = 0;
```

+ Theo lớp: CRectangle::count = 1;

7.3 Phương thức thành viên tĩnh (Hàm thành phần tĩnh)

- Được dùng chung cho tất cả các đối tượng của lớp

- Có thể được gọi mà không cần tạo ra đối tượng

- Chỉ có thể truy xuất thành viên tĩnh

- Phương thức không tĩnh (non-static) có thể truy xuất thành viên dữ liệu tĩnh

- Khai báo: **static return_type func (ds tham số);**

Ví dụ 1:

```
3 class Trainee
4 {
5 private:
6     int id;
7     char *name;
8     static int totalTrainees;
9 public:
10    Trainee(void);
11    Trainee(int _id, char *n);
12    static int getTotalTrainees();
13    ~Trainee(void);
14    int getID() const;
15    char* getName() const;
16    void setID(int _id);
17    void setName(char *n);
18 };
19
```

Khai báo và định nghĩa phương thức static

```
10 int Trainee::totalTrainees = 0;
11 Trainee::Trainee(int _id, char *n )
12 {
13     id = _id;
14     name = new char[strlen(n)+1];
15     strcpy(name,n);
16     totalTrainees++;
17 }
18 int Trainee::getTotalTrainees()
19 {
20     return totalTrainees;
21 }
22 Trainee::~Trainee(void)
23 {
24     if(name != NULL)
25         delete []name;
26     totalTrainees--;
27 }
```

Khởi tạo thành phần dữ liệu tĩnh

```
1 #include <iostream>
2 #include "Trainee.h"
3 using namespace std;
4
5 int main()
6 {
7     Trainee *t1, *t2;
8     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
9     t1 = new Trainee(1,"Lena");
10    cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
11    t2 = new Trainee(2,"Anna");
12    cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
13    delete t1;
14    cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
15    delete t2;
16    cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
17    return 0;
18 }
```

Kết quả:

```
So luong Trainee hien co: 0
So luong Trainee hien co: 1
So luong Trainee hien co: 2
So luong Trainee hien co: 1
So luong Trainee hien co: 0
Press any key to continue . . . _
```

Ví dụ 2:

```
#include <iostream>
using namespace std;
class Student {
private:
    string name;
    int age;
    double gpa;
    static int numberOfStudents;
public:
    Student(string _name, int _age, double _gpa);
```

```

    void setName(string _name);
    void setAge(int _age);
    void setGpa(double _gpa);
    string getName();
    int getAge();
    double getGpa();
    void display();
    static int getNumOfSt();
};

int Student::numberOfStudents = 0;
Student::Student(string _name, int _age, double _gpa) {
    name = _name;
    age = _age;
    gpa = _gpa;
    numberOfStudents++;
}

void Student::setName(string _name) {
    name = _name;
}

void Student::setAge(int _age)
{
    age = _age;
}

void Student::setGpa(double _gpa)
{
    gpa = _gpa;
}

string Student::getName()
{
    return name;
}

int Student::getAge()
{
    return age;
}

double Student::getGpa()
{
    return gpa;
}

int Student::getNumOfSt()
{
    return numberOfStudents;
}

int main() {
    cout << "Number of students: " << Student::getNumOfSt();
    Student student1("Hoang", 5, 4.0);
    cout << "\nNumber of students: " << Student::getNumOfSt();
    Student student2("Phong", 6, 3.7);
    cout << "\nNumber of students: " << Student::getNumOfSt();
    cout << "\nNumber of student1: " << student1.getNumOfSt(); //Dùng chung biến static
    return 0;
}

```

Kết quả:

Number of students: 0
 Number of students: 1
 Number of students: 2
 Number of student1: 2

8. Thành phần hằng (const)

8.1 Thành viên dữ liệu const

- Thành viên dữ liệu const sẽ không thay đổi giá trị trong suốt thời gian sống của đối tượng và gắn với đối tượng.
- Các object khác nhau sẽ có giá trị thành viên dữ liệu const khác nhau
- Được khởi tạo giá trị trong hàm tạo
- Khai báo: `const datatype cdata1, cdata2;`
- Khởi tạo:

```
class_name(ds_tham_số): cdata1(đoi so), cdata2(đoi so),...  
{  
    .....  
}
```

8.2 Phương thức thành viên const

- Mục đích: Ngăn chặn sự thay đổi của thành viên dữ liệu bên trong phương thức thành viên
- Phương thức get thường được khai báo với const
- Khai báo: `return_type Func(ds tham số) const;`

Ví dụ:

```
1 #include "Trainee_List.h"  
2  
3 Trainee_List::Trainee_List(int s):size(s)  
4 {  
5     //.....  
6 }  
7  
8 int Trainee_List::getSize() const  
9 {  
10     return size;  
11 }  
  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Khởi tạo thành viên dữ liệu const

```
6 int main()  
7 {  
8     //demo constant data member of class  
9     Trainee_List list1(100);  
10    Trainee_List list2(150);  
11    cout<<"List1 has "<<list1.getSize()<<" trainees\n";  
12    cout<<"List2 has "<<list2.getSize()<<" trainees\n";  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

id = id + 1;

error C2166: l-value specifies const object

8.3 Đối tượng hằng (const object)

- Là đối tượng *không thể thay đổi giá trị của dữ liệu thành viên*

- Đối tượng hằng *chỉ có thể gọi phương thức thành viên là tĩnh (static) hoặc hằng (const)*

- Khai báo:

const class_name obj;

Ví dụ:

```
5 int main()
6 {
7     //demo const object
8     const Trainee cTrainee(3, "Maria");
9
10    cTrainee.setID(4); //thay đổi giá trị thành viên dữ liệu --> error
11    cTrainee.printStandard(); //non-const function - error
12
13    cTrainee.getTotalTrainees(); //static function - ok
14    cout<<"Name: "<<cTrainee.getName()<<endl; //const function- ok
15
16    //demo phương thức static
17    Trainee *t1, *t2;
18    cout<<"Số lượng Trainee hiện có: "<<Trainee::getTotalTrainees()<<endl;
19    t1 = new Trainee(1, "Lena");
20    cout<<"Số lượng Trainee hiện có: "<<Trainee::getTotalTrainees()<<endl;
```

```
nh so 1\exp2\main.cpp(9) : error C2662: 'Trainee::setID' : cannot convert 'this' pointer from 'const Trainee' to 'Trainee &'
ifiers
nh so 1\exp2\main.cpp(10) : error C2662: 'Trainee::printStandard' : cannot convert 'this' pointer from 'const Trainee' to
```

9. Con trỏ this

- Con trỏ **this** là một **con trỏ hằng** được chương trình **tự định nghĩa** bên trong một phương thức, nó sẽ giữ và chỉ có thể **giữ địa chỉ của đối tượng đang gọi thực hiện phương thức đó**. Vì thế, con trỏ **this** luôn có kiểu trùng với kiểu của lớp đối tượng mà nó thuộc về.

- Một ví dụ về việc gọi phương thức:

```
hs.Nhap(); // đối tượng hs gọi thực hiện phương thức Nhap
```

Khi dòng lệnh trên được thực hiện, chương trình sẽ tự động gán địa chỉ của đối tượng hs vào biến con trỏ **this** (được định nghĩa ngầm bên trong phương thức Nhap). Sau đó, ta có thể dùng con trỏ này để truy cập đến các thuộc tính của đối tượng hs, cũng như dùng nó để gọi các phương thức khác, ví dụ:

```
void HocSinh::Nhap() {
    // ...
    cin >> this->mssv; // nhập MSSV của hs // ...
    this->XuLy(); // gọi XuLy() trên đối tượng hs
}
```

- Tuy nhiên để cho gọn, chương trình cho phép ta truy cập trực tiếp đến các thành viên của đối tượng đang gọi thực hiện phương thức. **Bất kì tên của thành viên nào được ghi ra mà không nói gì thêm thì thành viên đó sẽ xem như là được truy cập thông qua con trỏ this.**

* **Đặc điểm:**

- Được sử dụng làm **tham số ngầm định** trong tất cả các hàm thành viên **non-static** của lớp
- Cho phép các đối tượng **tham chiếu đến chính nó** thông qua “this”
- Trỏ đến **đối tượng** hiện thời đang gọi phương thức thành viên
- Con trỏ this **không thể được sử dụng trong phương thức tĩnh**
- Tránh sự mơ hồ:

Ví dụ 1:

```
#include <iostream>
using namespace std;

class Circle {
```

```
void Circle::setRadius(int radius)
{
    this->radius = radius;
}
```

<pre>private: int radius; public: /*Circle(void); ~Circle(void);*/ void setRadius(int radius); int getRadius() const; };</pre>	<pre>int Circle::getRadius() const { return this->radius; } int main() { Circle c1; c1.setRadius(2); cout << "Radius of Circle: " << c1.getRadius() << endl; return 0; }</pre>
--	--

Kết quả: Radius of Circle: 2

- Phương thức trả về đối tượng

Ví dụ 2:

<pre>#include <iostream> using namespace std; class Circle { private: int radius; const float PI; public: Circle(void); //~Circle(void); void setRadius(int radius); int getRadius() const; Circle scaleUp(int iTimes); float area(); };</pre>	<pre>float Circle::area() { return radius * radius * PI; } Circle::Circle(void):PI(3.14){ } void Circle::setRadius(int radius) { this->radius = radius; } int Circle::getRadius() const { return this->radius; } Circle Circle::scaleUp(int iTimes) { this->radius = radius * iTimes; return (*this); } int main() { Circle c1; c1.setRadius(2); cout << "Area of Circle1: " << c1.area() << endl; Circle c2 = c1.scaleUp(2); cout << "Area of Circle2: " << c2.area() << endl; return 0; }</pre>
---	---

Kết quả:

Area of Circle1: 12.56

Area of Circle2: 50.24

10. Hàm bạn, lớp bạn

10.1 Friend

- Giả sử có lớp Vector, lớp Matrix. Cần viết hàm **nhân** Vector với một Matrix. Hàm **nhân**:

+ Không thể thuộc lớp Vector hay lớp Matrix.

+ Không thể tự do

- Dữ liệu được khai báo “private” trong lớp không thể được truy xuất từ bên ngoài. Tuy nhiên, trong một số trường hợp các hàm/lớp bên ngoài muốn truy xuất dữ liệu “private”

→ C++ hỗ trợ một ngoại lệ: **“friend”**, cho phép khai báo **“friend”** với hàm/lớp

- **Hàm bạn không thuộc lớp**. Tuy nhiên, có quyền truy cập các thành viên **private** của lớp.

- Khi định nghĩa một lớp, có thể khai báo một hay nhiều hàm “bạn” (bên ngoài lớp)

- **Ưu điểm:** Kiểm soát các truy nhập ở cấp độ lớp – không thể áp đặt hàm bạn cho lớp nếu điều đó không được dự trù trước trong khai báo của lớp.

- Đây là cách cho phép chia sẻ dữ liệu giữa các đối tượng với một hàm tùy ý trong chương trình (**hàm friend**) hoặc chia sẻ các thành phần của đối tượng có thuộc tính **private** hay **protected** với các đối tượng khác (**lớp friend**).

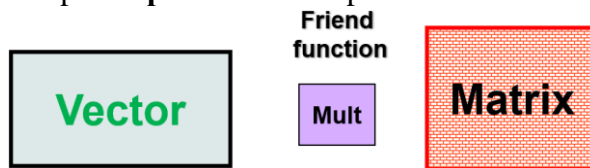
Khi một hàm mang thuộc tính friend cho phép chúng ta coi hàm này như là bạn của lớp, và đương nhiên đã là bạn bè cũng nên có một chút ưu đãi khi “bước vào nhà”

* **Tính chất của quan hệ “friend”**

- Được cho, không được nhận (Lớp B là bạn của lớp A, lớp A phải khai báo rõ ràng B là bạn của nó)
- Không đối xứng (nếu B là bạn của A thì A không nhất thiết phải là bạn của B)
- Không bắc cầu (nếu A là bạn của B, B là bạn của C, thì A không nhất thiết là bạn của C)
- Quan hệ friend có vẻ như vi phạm khái niệm đóng gói (encapsulation) của OOP nhưng có khi lại cần đến nó để cài đặt các mối quan hệ giữa các lớp và khả năng **đa năng hóa toán tử** trên lớp.

10.2 Hàm bạn

- Hàm bạn của một lớp **không phải là hàm thành viên của lớp đó**
- Được phép truy xuất đến các thành phần “**private**” của lớp



- **Tính chất của hàm bạn:**

- + Khai báo **nguyên mẫu hàm bên trong khai báo lớp** với từ khóa friend
- + Được định nghĩa **bên ngoài phạm vi lớp**
- + Được gọi giống như **hàm không thành viên**

- **Khai báo:**

friend <returntype> <func_name>(list_param);

- **Các kiểu bạn bè:**

- + Hàm tự do là bạn của một lớp
- + Hàm thành viên của một lớp là bạn của một lớp khác
- + Hàm bạn của nhiều lớp

- **Sử dụng hàm bạn trong trường hợp:**

- + Cần có một hàm có thể truy xuất thành viên “private” của hai hoặc nhiều lớp khác nhau.
- + Tạo ra các hàm xuất/nhập
- + Thiết kế một vài toán tử

Ví dụ 1:

<pre>#include <iostream> using namespace std; class Point { private: int x, y; public: Point(int x = 0, int y = 0) { this->x = x; this->y = y; }; friend bool same(Point p1, Point p2); //~Point(void); }; int main() { Point p1(3, 0), p2(3), p3; if (same(p1, p2)) cout << "p1 trung voi p2." << endl; else cout << "p1 khac p2."; if (same(p1, p3)) cout << "p1 trung voi p3." << endl; else cout << "p1 khac p3.";</pre>	<p>Kết quả: p1 trung voi p2. p1 khac p3.</p>
--	---

<pre> return 0; } //Đn hàm ngoài pvi lớp Point bool same(Point p1, Point p2) { return ((p1.x == p2.x) && (p1.y == p2.y)); } </pre>	
---	--

Ví dụ 2:

```

class Matrix;

class Vector {
    friend Vector multiply(const Matrix&, const Vector&);
    ... }

class Matrix {
    friend Vector multiply(const Matrix&, const Vector&);
    ... }

Vector multiply(const Matrix& m1, const Vector& v1) {
    ... }

```

10.3 Lớp bạn

- **Friend class:** tất cả các hàm thành viên của lớp A có thể truy xuất dữ liệu “private” của lớp B → A là bạn của B

Ví dụ 1:

```

class Point {
    ...
    friend class Line;
};

```

Ví dụ 2:

<pre> #include <iostream> using namespace std; class Square; class Rectangle { int width, height; public: int area() { return (width * height); } void convert(Square a); }; class Square { friend class Rectangle; private: int side; public: Square(int a) : side(a) {} }; </pre>	<pre> void Rectangle::convert(Square a) { width = a.side; height = a.side; } int main() { Rectangle rect; Square sqr(4); rect.convert(sqr); cout << rect.area(); return 0; } </pre> <p>Kết quả: 16</p>
--	--

Ví dụ 3:

```
#include <iostream>
using namespace std;

class A {
    int x = 5;
    friend class B; // lớp friend
};

class B {
public:
    void display(A& a) {
        cout << "Gia tri cua x la: " << a.x;
    }
};

int main() {
    A a;
    B b;
    b.display(a);
    return 0;
}
```

Kết quả:

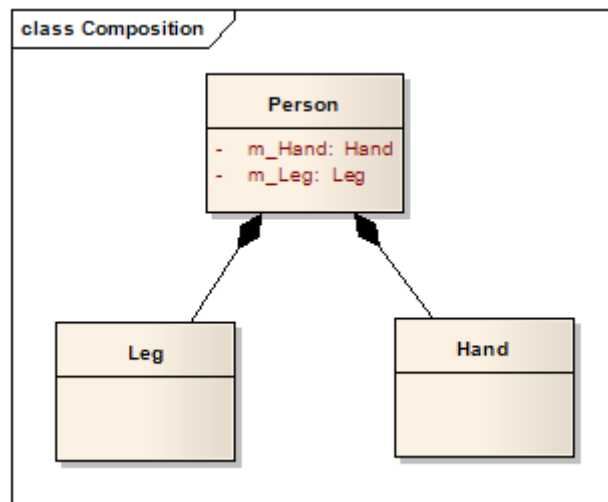
Gia tri cua x la: 5

* Nhận xét:

- Hàm/lớp bạn phá vỡ tính đóng gói, bảo mật dữ liệu
- Chỉ nên sử dụng khi thực sự cần thiết

11. Đối tượng là thành phần của lớp

- **Đối tượng có thể là thành phần của đối tượng khác**, khi một đối tượng thuộc lớp “lớn” được tạo ra, các thành phần của nó cũng được tạo ra.



- **Phương thức thiết lập (constructor) (nếu có) sẽ được tự động gọi cho các đối tượng thành phần.**
- **Khi đối tượng kết hợp bị hủy → đối tượng thành phần của nó cũng bị hủy**, nghĩa là phương thức hủy bỏ sẽ được gọi cho các đối tượng thành phần, sau khi phương thức hủy bỏ của đối tượng kết hợp được gọi.
- Nếu đối tượng thành phần phải cung cấp tham số khi thiết lập thì đối tượng kết hợp (đối tượng lớn) **phải có phương thức thiết lập** để cung cấp tham số thiết lập cho các đối tượng thành phần.
- **Cú pháp để khởi động đối tượng thành phần là dùng dấu hai chấm (:)** theo sau bởi tên thành phần và tham số khởi động.

```

#include <iostream>
using namespace std;

class Diem {
    double x, y;
public:
    Diem(double xx = 0, double yy = 0) : x(xx), y(yy) {
    }
    void Set(double xx, double yy) {
        x = xx;
        y = yy;
    }
    double getX() { return x; }
    double getY() { return y; }
};

class TamGiac {
    Diem A, B, C;
    int loai;
public:
    //Cú pháp dấu hai chấm cũng được dùng cho đối tượng thành phần thuộc kiểu cơ sở
    TamGiac(double xA, double yA, double xB, double yB, double xC, double yC)
        : A(xA, yA), B(xB, yB), C(xC, yC) {
    }
    void getA() { cout << A.getX() << ", " << A.getY(); }
    void getB() { cout << B.getX() << ", " << B.getY(); }
    void getC() { cout << C.getX() << ", " << C.getY(); }
};

int main() {
    TamGiac t(100, 100, 200, 400, 300, 300);
    cout << "Toa do diem A: "; t.getA(); cout << endl;
    cout << "Toa do diem B: "; t.getB(); cout << endl;
    cout << "Toa do diem C: "; t.getC(); cout << endl;
    return 0;
}

```

Toa do diem A: 100, 100

Toa do diem B: 200, 400

Toa do diem C: 300, 300

12. Đối tượng là thành phần của mảng

- Khi một mảng được tạo ra → các phần tử của nó cũng được tạo ra → phương thức thiết lập sẽ được gọi cho từng phần tử.

- Vì không thể cung cấp tham số khởi động cho tất cả các phần tử của mảng → khi khai báo mảng, mỗi đối tượng trong mảng phải có **khả năng tự khởi động**, nghĩa là có thể thiết lập không cần tham số.

- Đối tượng có khả năng tự khởi động trong những trường hợp nào?

- + Lớp không có phương thức thiết lập
- + Lớp có phương thức thiết lập không tham số
- + Lớp có phương thức thiết lập mà mọi tham số đều có giá trị mặc nhiên

<pre>class Diem { double x, y; public: Diem(double xx = 0, double yy = 0) : x(xx), y(yy) { } void Set(double xx, double yy) { x = xx; y = yy; } double getX() { return x; } double getY() { return y; } };</pre>	<pre>class String { char* p; public: String(char* s) { p = _strdup(s); } String(const String& s) { p = _strdup(s.p); } ~String() { cout << "delete " << (void*)p << "\n"; delete[] p; } };</pre>
<pre>class SinhVien { String MaSo; String HoTen; int NamSinh; public: SinhVien(char* ht, char* ms, int ns) : HoTen(ht), MaSo(ms), NamSinh(ns) { } };</pre>	<pre>String arrs[3]; //Error Diem arrd[5]; //Error SinhVien arrsv[7]; //Error</pre>

* Dùng phương thức thiết lập với tham số có giá trị mặc nhiên (Class Diem giữ nguyên)

```
class String {
    char* p;
public:
    String(char* s) { p = strdup(s); }
    String() { p = strdup(""); }
    ~String() {
        cout << "delete " << (void*)p << "\n";
        delete[] p;
    }
};

class SinhVien {
    String MaSo, HoTen;
    int NamSinh;
public:
    SinhVien(char* ht = "Nguyen Van A", char* ms = "19920014", int ns = 1982) : HoTen(ht),
        MaSo(ms), NamSinh(ns) { }
};

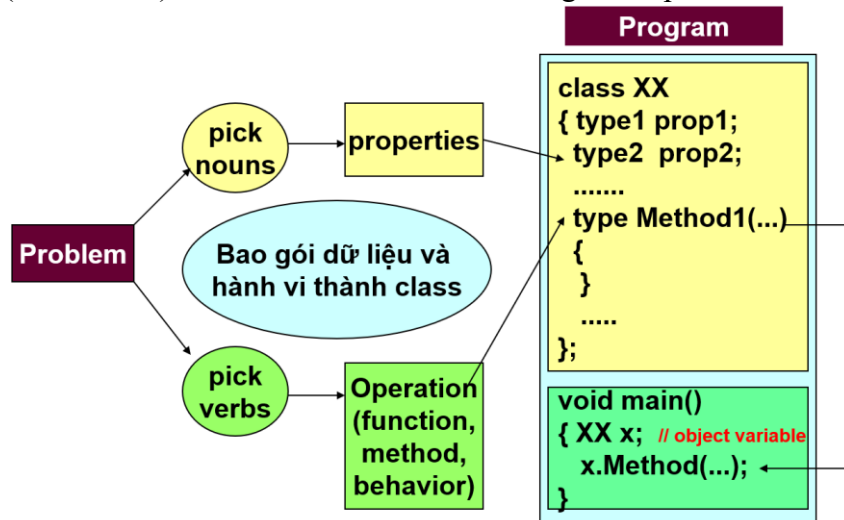
String as[3]; // OK: Ca ba phan tu deu la chuoi rong
Diem ad[5]; // OK: ca 5 diem deu la (0,0)
SinhVien asv[7]; // OK: ca 7 sinh vien deu co cung hoten, maso, namsinh
```

* Dùng phương thức thiết lập không tham số

<pre>class Diem { double x, y; public: Diem(double xx, double yy) : x(xx), y(yy) { } Diem() : x(0), y(0) { } // ... };</pre>	<pre>class String { char* p; public: String(char* s) { p = strdup(s); } String() { p = strdup(""); } ~String() { cout << "delete " << (void*)p << "\n"; delete[] p; } };</pre>
<pre>class SinhVien { String MaSo, HoTen; int NamSinh; public: SinhVien(char* ht, char* ms, int ns) : HoTen(ht), MaSo(ms), NamSinh(ns) { } SinhVien() : HoTen("Nguyen Van A"), MaSo("19920014"), NamSinh(1982) { } }; String as[3]; // Ca ba phan tu deu la chuoi rong Diem ad[5]; // ca 5 diem deu la (0,0) SinhVien asv[7]; // Ca 7 sinh vien deu co cung hoten, maso, namsinh</pre>	

13. Các nguyên tắc xây dựng lớp

- **Hình thành lớp:** Khi ta nghĩ đến “nó” như một khái niệm riêng lẻ → Xây dựng lớp biểu diễn khái niệm đó.
- **Lớp** là biểu diễn cụ thể của một khái niệm vì vậy **tên lớp luôn là danh từ**.
- **Các thuộc tính** của lớp là các thành phần dữ liệu nên chúng **luôn là danh từ**.
- Các thuộc tính dữ liệu phải vừa đủ để mô tả khái niệm, không dư, không thiếu.
- **Các hàm thành phần** (các hành vi) là các thao tác chỉ rõ hoạt động của lớp nên **các hàm là động từ**.



- Các thuộc tính **có thể suy diễn từ những thuộc tính khác** thì dùng hàm thành phần để thực hiện tính toán. Ví dụ chu vi, diện tích của một tam giác

<pre>class TamGiac{ Diem A,B,C; double ChuVi; double DienTich; public: //... };</pre>	<pre>class TamGiac{ Diem A,B,C; public: //... double ChuVi() const; double DienTich() const; };</pre>
---	---

- Nên khai báo hằng đối với:

- + Các đối tượng mà ta không định sửa đổi. Ví dụ: `const double PI = 3.14;`
- + Các tham số của hàm mà ta không định cho hàm đó sửa đổi: `void printHeight(const LargeObj &LO) { cout << LO.height; }`
- + Các hàm thành viên không thay đổi đối tượng chủ: `int Date::getDay() const { return day; }`

- Tuy nhiên, nếu các thuộc tính suy diễn đòi hỏi nhiều tài nguyên hoặc thời gian để thực hiện tính toán, ta có thể khai báo là dữ liệu thành phần.

```
class QuocGia{
    long DanSo;
    double DienTich;
    double TuổiTrungBinh;
public:
    double TínhTuoiTB() const;
    //...
};
```

- Dữ liệu thành phần nên được kết hợp thay vì phân rã:

<pre>class TamGiac{ Diem A,B,C; public: //... }; class HìnhTron{ Diem Tam; double BanKinh; public: //... };</pre>	<pre>class TamGiac{ double xA, yA; double xB, yB, xC, yC; public: //... }; class HìnhTron{ double tx, ty, BanKinh; public: //... };</pre>
---	---

- Trong mọi trường hợp, nên có phương thức thiết lập (Constructor) để khởi động đối tượng
- Nên có phương thức thiết lập có khả năng tự khởi động không cần tham số
- Nếu đối tượng có nhu cầu cấp phát tài nguyên thì phải có phương thức thiết lập, copy constructor để khởi động đối tượng bằng đối tượng cùng kiểu và có destructor để dọn dẹp. Ngoài ra còn có phép gán.
- Nếu đối tượng đơn giản không cần tài nguyên riêng → Không cần copy constructor và destructor