

NGÔN NGỮ LẬP TRÌNH C++

Chương 5: Lớp



Nội dung

- Đặc điểm của lập trình hướng đối tượng
- Khái niệm lớp, đối tượng
- Các thành phần của lớp
- Phạm vi truy cập lớp
- Hàm khởi tạo và hàm hủy
- Con trỏ và mảng các đối tượng



Đặc điểm của lập trình hướng đối tượng

- Tập trung vào dữ liệu thay cho các hàm.
- Chương trình được chia thành các đối tượng.
- Các cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng.
- Các hàm xác định trên các vùng dữ liệu của đối tượng được gắn với nhau trên cấu trúc dữ liệu đó.
- Dữ liệu được bao bọc, che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
- Các đối tượng trao đổi với nhau thông qua các hàm.
- Dữ liệu và các hàm mới có thể dễ dàng bổ sung vào đối tượng nào đó khi cần thiết.
- Chương trình được thiết kế theo cách tiếp cận bottom-up (dưới-lên).



Đối tượng (Object)

- Đối tượng là một thực thể cụ thể



Lớp

- Lớp là khái niệm trung tâm của lập trình hướng đối tượng, nó là sự mở rộng của các khái niệm cấu trúc (struct) của C và bản ghi (record) của PASCAL.
- Giống như cấu trúc, lớp có thể xem như một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và lớp được dùng để khai báo các biến, mảng đối tượng
- Một lớp là một thiết kế (blueprint) hay mẫu (prototype) cho các đối tượng cùng kiểu
 - Ví dụ: TAO, LE, BUOI, CAM là các loại quả trong lớp HOA_QUA



Lớp và đối tượng

- C++ coi lớp là sự trừu tượng hóa các đối tượng, là một khuôn mẫu để biểu diễn các đối tượng thông qua các thuộc tính và các hành động đặc trưng của đối tượng.
- Để định nghĩa một lớp trong C++, ta dùng từ khóa class với cú pháp:

```
class <Tên lớp>{  
};
```

- Ví dụ

```
class Point{  
};
```

- **Nên** đặt tên Class có ký tự bắt đầu mỗi từ bằng chữ in hoa.



Sử dụng lớp đối tượng

- Lớp đối tượng được sử dụng khi ta khai báo các thể hiện của lớp đó. Một thể hiện của một lớp chính là một đối tượng cụ thể của lớp đó. Việc khai báo một thể hiện của một lớp được thực hiện như cú pháp khai báo một biến có kiểu lớp:

<Tên lớp> <Tên biến lớp>;

- Trong đó:
 - Tên lớp: là tên lớp đối tượng đã được định nghĩa trước khi khai báo biến.
 - Tên biến lớp: là tên đối tượng cụ thể. Tên biến lớp sẽ được sử dụng như các biến thông thường trong C++, ngoại trừ việc nó có kiểu lớp.

- Ví dụ

Point myPoint;

Point p[10];



Các thành phần của lớp

- Việc khai báo các thành phần của lớp có dạng như sau:

```
class <Tên lớp>{
```

```
    private:
```

```
        <Khai báo các thành phần riêng>
```

```
    protected:
```

```
        <Khai báo các thành phần được bảo vệ>
```

```
    public:
```

```
        <Khai báo các thành phần công khai>
```

```
};
```



Các thành phần của lớp

- Các thành phần của lớp được chia làm hai loại:
- 1. Các thành phần chỉ dữ liệu của lớp, được gọi là thuộc tính của lớp
- 2. Các thành phần chỉ hành động của lớp, được gọi là phương thức của lớp



Thuộc tính của lớp

- Thuộc tính của lớp là thành phần chứa dữ liệu, đặc trưng cho các tính chất của lớp. Thuộc tính của lớp được khai báo theo cú pháp sau:
 <Kiểu dữ liệu> <Tên thuộc tính>;
- Kiểu dữ liệu: có thể là các kiểu dữ liệu cơ bản của C++, cũng có thể là các kiểu dữ liệu phức tạp do người dùng tự định nghĩa như struct, hoặc kiểu là một lớp đã được định nghĩa trước đó.
- Tên thuộc tính: là tên thuộc tính của lớp, có tính chất như một biến thông thường. Tên thuộc tính phải tuân theo quy tắc đặt tên biến của C++.
- Ví dụ

```
class Point {  
    private:  
        int x,y; ;//toa do x va toa do y  
};
```



Thuộc tính của lớp

Truy cập thuộc tính của lớp

- Nếu thuộc tính được dùng bên ngoài phạm vi lớp, cú pháp phải thông qua tên biến lớp (cách này chỉ sử dụng được với các biến có tính chất public):
 <Tên biến lớp>.<tên thuộc tính>;
- Nếu thuộc tính được dùng bên trong lớp, cú pháp đơn giản hơn:
 <Tên thuộc tính>;



Phương thức của lớp

- Trong C++, việc cài đặt chi tiết nội dung của phương thức có thể tiến hành ngay trong phạm vi lớp hoặc bên ngoài phạm vi định nghĩa lớp. Cú pháp chỉ khác nhau ở dòng khai báo tên phương thức.
- Nếu cài đặt phương thức ngay trong phạm vi định nghĩa lớp, cú pháp là:
 <Kiểu trả về> <Tên phương thức>([<Các tham số>]){
 // Cài đặt chi tiết
 }
- Nếu cài đặt phương thức bên ngoài phạm vi định nghĩa lớp, ta phải dùng chỉ thị phạm vi "::" để chỉ ra rằng đây là một phương thức của lớp mà không phải là một hàm tự do trong chương trình:
 <Kiểu trả về> <Tên lớp>::<Tên phương thức>([<Các tham số>]){
 ... // Cài đặt chi tiết
 }



Phương thức của lớp

Truy cập phương thức của lớp

- Nếu phương thức được dùng bên ngoài phạm vi lớp, cú pháp phải thông qua tên biến lớp (cách này chỉ sử dụng được với các phương thức có tính chất public):

<Tên biến lớp>.<Tên phương thức>([<Các tham số>]);

- Nếu phương thức được dùng bên trong lớp, cú pháp đơn giản hơn:

<Tên phương thức>([<Các tham số>]);



Phạm vi truy cập lớp

Trong C++, có một số khái niệm về phạm vi, xếp từ bé đến lớn như sau:

- **Phạm vi khối lệnh:** Trong phạm vi giữa hai dấu giới hạn “{}” của một khối lệnh. Ví dụ các lệnh trong khối lệnh lặp while(){} sẽ có cùng phạm vi khối lệnh.
- **Phạm vi hàm:** Các lệnh trong cùng một hàm có cùng mức phạm vi hàm.
- **Phạm vi lớp:** Các thành phần của cùng một lớp có cùng phạm vi lớp với nhau: các thuộc tính và các phương thức của cùng một lớp.
- **Phạm vi chương trình:** Các lớp, các hàm, các biến được khai báo và định nghĩa trong cùng một tệp chương trình thì có cùng phạm vi chương trình.



Phạm vi truy cập lớp

- Trong phạm vi truy nhập lớp, ta chỉ quan tâm đến hai phạm vi lớn nhất, đó là phạm vi lớp và phạm vi chương trình.
- Trong C++, phạm vi truy nhập lớp được quy định bởi các từ khóa về thuộc tính truy nhập:
 - private: Các thành phần có thuộc tính private thì sẽ được giấu kín, bảo vệ an toàn dữ liệu của lớp, không cho phép các hàm bên ngoài xâm nhập vào dữ liệu của lớp, chỉ có thể được truy nhập trong phạm vi lớp.
 - protected: Các thành phần lớp có thuộc tính protected chỉ có thể được truy nhập trong phạm vi lớp. Ngoài ra nó còn có thể được truy nhập trong các lớp con khi có kế thừa
 - public: các thành phần lớp có thuộc tính public thì có thể được truy nhập trong phạm vi chương trình, có nghĩa là nó có thể được truy nhập trong các hàm tự do, các phương thức bên trong các lớp khác



Bài tập

- Xây dựng lớp Phân số và các phương thức của lớp phân số
 - Phương thức Nhập, Xuất
 - Phương thức cộng, trừ hai phân số



Hàm bạn

- Xét ví dụ sau:

```
class Diem {  
    private:  
        int x,y;//toa do x va toa do y  
    public:  
        void nhapd();//phuong thuc nhap  
};
```



Hàm bạn

```
float KC2D(Diem D1, Diem D2){ //hàm tính khoảng cách giữa 2 điểm  
    return sqrt((D1.x-D2.x)*(D1.x-D2.x)+(D1.y-D2.y)*(D1.y-D2.y));  
}
```

Thấy xuất hiện lỗi không cho phép truy cập thuộc tính x và y.

Nguyên nhân: x, y là thành phần private, một hàm thông thường hay tại hàm chính không truy cập trực tiếp.

Giải pháp:

- + Thay đổi thuộc tính của x,y là public (không tốt)
- + Khai báo hàm KC2D dưới danh nghĩa hàm bạn của lớp



Hàm bạn

- Hàm bạn của lớp là hàm được khai báo bên trong lớp bắt đầu bằng từ khóa **friend**. Được xây dựng bên ngoài lớp. Được phép truy cập vào bất cứ thành viên nào của lớp mà nó nhận là bạn.
- Một hàm có thể là bạn của nhiều lớp.
- Tất cả các hàm của một lớp là bạn của lớp khác (lớp bạn)



Hàm bạn

- Xây dựng hàm bạn

```
class <Tên lớp>{  
    // Khai báo các thành phần lớp như thông thường  
    // Khai báo hàm bạn  
    friend <Kiểu trả về> <Tên hàm bạn>([<Các tham số>]);  
};  
  
<Kiểu trả về> <Tên hàm bạn>([<Các tham số>]){  
    // Có thể truy nhập trực tiếp các thành phần private của lớp đã khai báo  
}
```



Hàm bạn

c.Ví dụ:

```
class Diem{  
    private:  
    int x,y;//toa do x va toa do y  
    public:  
    void nhapd();//phuong thuc nhap  
    friend float KC2D(Diem D1, Diem D2)  
};  
  
float KC2D(Diem D1, Diem D2){  
    return sqrt((D1.x-D2.x)*(D1.x-D2.x)+(D1.y-D2.y)*(D1.y-D2.y));  
}
```



Hàm bạn

- Hàm bạn không phải là một phương thức của lớp. Nó là hàm tự do, việc định nghĩa và sử dụng hàm này hoàn toàn tương tự như các hàm tự do khác.
- Việc khai báo khuôn mẫu hàm bạn trong phạm vi lớp ở vị trí nào cũng được: hàm bạn không bị ảnh hưởng bởi các từ khóa private, protected hay public trong lớp.
- Trong hàm bạn, có thể truy nhập trực tiếp đến các thành phần private và protected của đối tượng có kiểu lớp mà nó làm bạn (truy nhập thông qua đối tượng cụ thể).



Lớp bạn

- **Nếu lớp A được khai báo là bạn của lớp B** thì tất cả các phương thức của A đều có thể truy nhập đến các thành phần riêng của lớp B. Một lớp có thể là bạn của nhiều lớp khác. Cũng có thể khai báo A là bạn của B và B là bạn của A.
- Một lớp có thể là bạn của vô số lớp.



Lớp bạn

Giả sử có 3 lớp A, B và C. Để khai báo lớp class B{ này là bạn của lớp kia, ta viết theo mẫu sau:

// Khai báo trước các lớp

```
class A;
```

```
class B ;
```

```
class C;
```

// Định nghĩa các lớp

```
class A {
```

```
...
```

```
friend class B ; // Lớp B là bạn của A
```

```
friend class C ; // Lớp C là bạn của A
```

```
...
```

```
};
```

```
...
```

```
friend class A ; // Lớp A là bạn của B
```

```
friend class C ; // Lớp C là bạn của B
```

```
...
```

```
};
```

```
class C
```

```
{
```

```
...
```

```
friend class B ; // Lớp B là bạn của C
```

```
...
```

```
};
```



Phương thức getter / setter

- Phương thức getter/setter được sử dụng để thao tác gán / truy cập giá trị tới thuộc tính private.
- Ví dụ

```
void setAge(int age) {  
    this->age = age;  
}  
int getAge() {  
    return this->age;  
}
```



Hàm tạo

- Là phương thức của lớp (nhưng khá đặc biệt) dùng để tạo dựng một đối tượng mới.
- Ngay cả khi không được khai báo tường minh, C++ cũng gọi hàm khởi tạo ngầm định khi đối tượng được khai báo.
- Hàm khởi tạo của một lớp được khai báo tường minh theo cú pháp sau:

```
class <Tên lớp>{  
    public:  
    <Tên lớp>([<Các tham số>]); // Khai báo hàm khởi tạo  
};
```



Hàm tạo

Nhận xét:

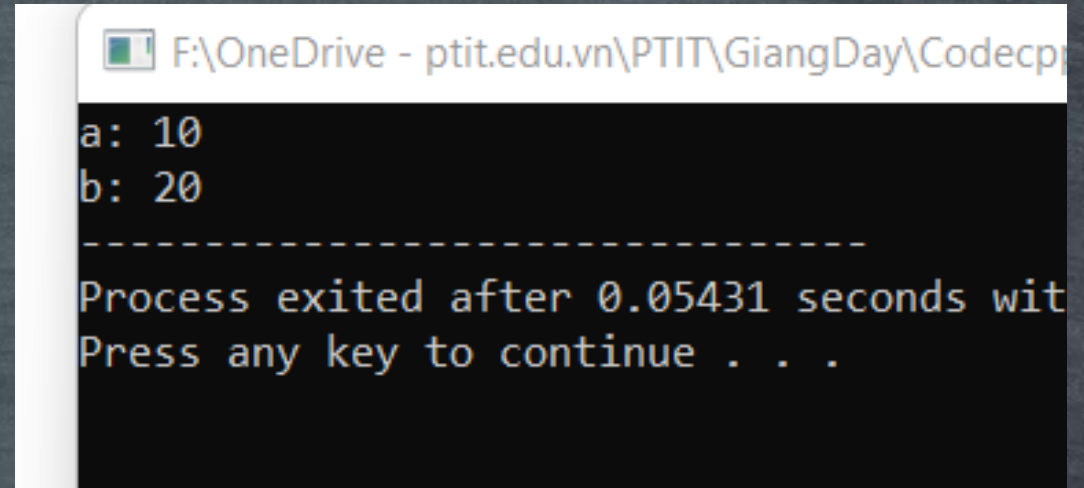
- Hàm tạo có **tên_hàm** trùng **Tên_lớp** không bắt đầu bằng từ khóa **void** hay **kiểu dữ liệu**.
- Hàm tạo phải được khai báo bên trong lớp
- Hàm tạo có thể được xây dựng bên trong hay bên ngoài lớp.
- Hàm tạo có hoặc không có tham số truyền vào.



Hàm tạo

- Ví dụ:

```
#include <iostream>
using namespace std;
class construct {
public:
    int a, b;
    construct() // Default Constructor
    {          a = 10;      b = 20;    }
};
int main(){
    // Default constructor called automatically when the object is created
    construct c;
    cout << "a: " << c.a << endl << "b: " << c.b;
    return 1;
}
```

A screenshot of a Windows command prompt window. The title bar shows the file path 'F:\OneDrive - ptit.edu.vn\PTIT\GiangDay\Codecp'. The command prompt displays the output of the C++ program: 'a: 10' followed by 'b: 20' on the next line. Below this, a dashed line separates the output from the process exit message: 'Process exited after 0.05431 seconds with return code 1' and 'Press any key to continue . . .'.

```
F:\OneDrive - ptit.edu.vn\PTIT\GiangDay\Codecp
a: 10
b: 20
-----
Process exited after 0.05431 seconds with return code 1
Press any key to continue . . .
```


Hàm hủy

- Hàm hủy là hàm hủy bỏ bộ nhớ của đối tượng được cấp phát bởi hàm tạo
- Hàm hủy được tự động gọi đến khi mà đối tượng được giải phóng khỏi bộ nhớ.
- Cú pháp khai báo hàm hủy như sau:

```
class <Tên lớp>{  
    public:  
        ~<Tên lớp>([<Các tham số>]); // Khai báo hàm hủy  
};
```



Hàm hủy

- Ví dụ:

```
#include<iostream>
using namespace std;
class Test{
public:
    Test(){
        cout<<"\n Constructor executed";
    }
    ~Test(){
        cout<<"\n Destructor executed";
    }
};
main()
{
    Test t,t1,t2,t3;
    return 0;
}
```

```
Constructor executed
Constructor executed
Constructor executed
Constructor executed
Destructor executed
Destructor executed
Destructor executed
Destructor executed
```

```
-----
Process exited after 0.05388 seconds with return value 0
Press any key to continue . . .
```


Hàm hủy

- Nhận xét:
 - Mỗi lớp có duy nhất một hàm hủy
 - Hàm hủy không có đối
 - Hàm hủy không cần được triệu gọi, sẽ tự động được gọi khi kết thúc chương trình hay đối tượng bị hủy.



Hàm tạo và hàm hủy

1. Các lớp không có thành viên là kiểu con trỏ hay mảng thì không nhất thiết phải xây dựng hàm tạo, hàm hủy. Chương trình C++ sẽ tự phát sinh hàm tạo, hàm hủy không đối số và không làm gì cả.
2. Các lớp có thành viên kiểu con trỏ, mảng nhất thiết phải xây dựng hàm tạo, hàm hủy để cấp phát (toán tử new) và hủy bộ nhớ (toán tử delete).



Bài tập

- Xây dựng lớp đa thức. Với các hàm tạo, hàm hủy, phương thức nhập, xuất, cộng, trừ hai đa thức.



Con trỏ và mảng các đối tượng

- Con trỏ đối tượng được khai báo tương tự như khai báo các con trỏ có kiểu thông thường:

`<Tên lớp> *<Tên con trỏ đối tượng>;`

- Ví dụ, muốn khai báo một con trỏ đối tượng có kiểu của lớp Car, ta khai báo như sau:

`Car *myCar;`

- Con trỏ đối tượng cũng cần phải cấp phát bộ nhớ hoặc trỏ vào một địa chỉ của một đối tượng lớp xác định trước khi được sử dụng. Cấp phát bộ nhớ cho con trỏ đối tượng cũng bằng thao tác new:
- `<Tên con trỏ đối tượng> = new <Tên lớp>([<Các tham số>])`



Con trỏ và mảng các đối tượng

- Khai báo mảng tĩnh các đối tượng

<Tên lớp> <Tên biến mảng>[<Số lượng đối tượng>;

- Ví dụ:

Car cars[10];

- Một mảng các đối tượng cũng có thể được khai báo và cấp phát động thông qua con trỏ đối tượng như sau:

<Tên lớp> *<Tên biến mảng động> = new <Tên lớp>[<Độ dài mảng>;

- Ví dụ:

Car *cars = new Car[10];



Bài tập

- Khai báo lớp Nhân viên có 2 thuộc tính Họ tên và Lương. Nhập vào danh sách n nhân viên, đưa ra nhân viên có mức lương cao nhất.



Bài tập

- Xây dựng lớp môn học bao gồm các thông tin: Mã MH, Tên MH, Số TC, Điểm CC, Điểm KT, Điểm Thi; Lớp sinh viên gồm các thông tin: Mã SV, Họ Tên, Lớp. Mỗi sinh viên sẽ học một số môn nhất định. Thực hiện các nhiệm vụ sau:
- Nhập N, M và các thông tin cho N môn học, M sinh viên từ bàn phím. Mỗi sinh viên nhập K môn học.
- Tính điểm trung bình các môn học của từng sinh viên theo trọng số Điểm CC 10%, Điểm KT 20%, Điểm Thi 70% .
- In ra danh sách sinh viên có điểm TBC ≥ 7.0



Lớp bao

- Một lớp có thuộc tính là đối tượng của lớp khác gọi là lớp bao.

Ví dụ:

```
class A{  
    private:  
        int a, b;  
        ...  
};  
class B{  
    private:  
        double x, y, z;  
        ...  
};
```

```
class C{  
    private:  
        int m, n;  
        A u;  
        B p, q;  
        ...  
};
```

Trong ví dụ trên thì:
C là lớp bao
A, B là các lớp thành phần (của C)



Lớp bao

- Lưu ý: Trong các phương thức của lớp bao không cho phép truy nhập trực tiếp đến các thuộc tính của các đối tượng của các lớp thành phần.
- Khi xây dựng hàm tạo của lớp bao, phải sử dụng các hàm tạo của lớp thành phần để khởi gán cho các đối tượng thành phần của lớp bao.
- Gọi đến phương thức của lớp thành phần giống như là lời gọi phương thức ở chương trình chính thông qua tên đối tượng.



Lớp bao

- Cú pháp khai báo hàm tạo:

tên_lớp(danh sách đối) : tên_đối_tượng(danh sách giá trị), ...
,tên_đối_tượng(danh sách giá trị)

{

// Các câu lệnh trong thân hàm tạo

}



Lớp bao

```
class A
{
private:
int a, b;
public:

A()
{
a=b=0;
}
A(int a1, int b1)
{
a = a1; b = b1;
}
...
};
```

```
class B
{
private:
double x, y, z;
public:
B()
{
x = y = z = 0.0 ;
}
B(double x1, double y1)
{
x = x1; y = y1; z = 0.0 ;
}
B(double x1, double y1, double z1)
{
x = x1; y = y1; z = z1 ;
}
...
};
```

```
class C
{
private:
int m, n;
A u, v;
B p, q, r;
public:
C(int m1, int n1, int a1, int b1,
double x1, double y1, double x2, double y2, double z2) :
u(), v(a1,b1), q(x1,y1), r(x2,y2,z2)
{
m = m1 ; n = n1;
}
};
```

Bài tập

Xây dựng lớp Điểm trong hệ toạ độ xOy: với hàm tạo, hàm hủy, Phương thức nhập xuất, tính khoảng cách hai điểm.

Xây dựng lớp Tam giác là lớp bao của lớp Điểm (Tạo bởi 3 điểm): với các hàm tạo, hàm hủy, Phương thức nhập xuất, tính chiều dài 3 cạnh, chu vi.



Bài tập

Xây dựng lớp Điểm trong hệ toạ độ xOy: với hàm tạo, hàm hủy, Phương thức nhập xuất, tính khoảng cách hai điểm.

Xây dựng lớp Hình chữ nhật là lớp bao của lớp Điểm (Tạo bởi 1 điểm và 2 cạnh): với các hàm tạo, hàm hủy, Phương thức nhập xuất, tính diện tích, chu vi.

