NGÔN NGỮ LẬP TRÌNH C++

SLIDE 6: Lớp và đối tượng

Giảng viên: Th.S Bùi Văn Kiên

NỘI DUNG

- 1. Khái niệm lớp, đối tượng
- 2. Các thành phần của lớp
- 3. Phạm vi truy cập lớp
- 4. Hàm khởi tạo và hàm huỷ
- 5. Con trỏ và mảng các đối tượng

1. Khái niệm lớp, 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 Car{
};
```

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

1. Khái niệm lớp, đối tượng

Sử dụng:

- 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 Car myCar;

2. 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>
};
```

2. 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

2. Các thành phần 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++.

2. Các thành phần của lớp

- Ví dụ

```
class Car {
        private:
        int speed;
};
```

2. Các thành phần 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>;

2. Các thành phần của lớp

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
  }

2. Các thành phần của lớp

Truy cập phương thức của lớp

- Ví dụ:

Car car;

car.show();

```
class Car{
    int speed;
    char mark[20];
    float price;

    void show();
};


void Car::show(){
    cout << "This is a " << mark << " having a speed of " << speed << endl;
}
```

2. Các thành phần của lớp

Phương thức của lớp

- Nếu cài đặt phương thức bên ngoài phạm vi định nghĩa lớp, ta phải dùng chỉ thị phạm vi "::" để chỉ ra rằng đấy 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

}

2. Các thành phần của lớp

Truy cập phương thức của lớp

- Cũng tương tự như các thuộc tính của lớp, các phương thức cũng có thể được sử dụng bên ngoài lớp thông qua tên biến lớp, hoặc có thể được dùng ngay trong lớp bởi các phương thức khác của lớp định nghĩa nó.
- 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 thuộc tính được dùng bên trong lớp, cú pháp đơn giản hơn:

<Tên phương thức>([<Các tham số>]);

3. 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.

3. 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 của lớp có thuộc tính private thì chỉ có thể được truy nhập trong phạm vi lớp.
  - **protected:** Trong cùng một lớp, thuộc tính protected cũng có ảnh hưởng tương tự như thuộc tính private, 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

3. Phạm vi truy cập lớp

- Trong C++, phạm vi truy nhập lớp được quy định bởi các từ khóa về thuộc tính truy nhập:
  - **private:**
  - **protected:**
  - **public:**

```
class Car{
    int speed;
    char mark[20];
    float price;
public:
    Car(int speedIn, char markIn[], float priceIn){
        speed = speedIn;
        strcpy(mark, markIn);
        price = priceIn;
    }
};
```

Các biến của Car có phạm vi truy cập là gì?

3. Phạm vi truy cập lớp

Con trỏ this

- Trong C++, con trỏ this là một con trỏ đặc biệt trong lập trình hướng đối tượng, dùng để trỏ đến đối tượng hiện tại đang gọi phương thức.
- Con trỏ this chỉ tồn tại bên trong các phương thức của lớp và có vai trò rất quan trọng trong việc thao tác với các thuộc tính và phương thức của đối tượng hiện tại.

3. Phạm vi truy cập lớp

Tính chất con trỏ this

- This là con trỏ ngầm định được khai báo tự động bên trong mỗi phương thức của lớp.
- this trỏ đến địa chỉ của đối tượng hiện tại, tức là đối tượng đang gọi phương thức.
- this đặc biệt hữu ích trong các trường hợp như phân biệt giữa các biến thành viên và tham số có cùng tên, hoặc khi chúng ta muốn trả về đối tượng hiện tại trong một phương thức (thường gặp trong các phương thức thiết lập - setter).

3. Phạm vi truy cập lớp

Con trỏ this trong hàm constructor

- Trong constructor, this->name = name; và this->age = age; được dùng để gán giá trị cho biến thành viên name và age từ các tham số cùng tên.

→ Sử dụng con trỏ "this" để phân biệt giữa biến thành viên và tham số

```
class Person {
private:
    string name;
    int age;

public:
    /// Constructor
    Person(string name, int age) {
        this->name = name;
        this->age = age;
    }
}
```

3. Phạm vi truy cập lớp

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;
}
```

4. Hàm khởi tạo và hàm hủy

Hàm khởi tạo

- Hàm khởi tạo được gọi mỗi khi khai báo một đối tượng của lớp. 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
};
```

4. Hàm khởi tạo và hàm hủy

Hàm khởi tạo

- Ví dụ hàm khởi tạo

```
class Car{
        int speed;
        char mark[20];
        float price;
public:
        Car(int speedIn, char markIn[], float priceIn){
                speed = speedIn;
                strcpy(mark, markIn);
                price = priceIn;
        }
};
```

4. Hàm khởi tạo và hàm hủy

Hàm hủy

- Hàm hủy được tự động gọi đến khi mà đối tượng được giải phóng khỏi bộ nhớ. Nhiệm vụ của hàm hủy là dọn dẹp bộ nhớ trước khi đối tượng bị giải phóng. Cú pháp khai báo hàm hủy như sau:

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

4. Hàm khởi tạo và hàm hủy

Hàm hủy

- Ví dụ hàm hủy

```
class Car{
        int speed;
        char *mark;
        float price;
public:
        ~Car(){
                delete [] mark;
        }
};
```

4. Hàm khởi tạo và hàm hủy

Hàm hủy

- Nếu không viết hàm hủy trong lớp (class) thì trình biên dịch sẽ tự động tạo ra một hàm hủy mặc định. Hàm hủy mặc định này có nhiệm vụ giải phóng tài nguyên mà đối tượng chiếm dụng khi nó bị hủy.
- Tuy nhiên, nếu lớp đó chứa con trỏ hoặc tài nguyên động (như bộ nhớ được cấp phát bằng new), bạn cần phải định nghĩa hàm hủy để đảm bảo tài nguyên được giải phóng đúng cách và tránh rò rỉ bộ nhớ.

(memory leak)

4. Hàm khởi tạo và hàm hủy

Ví dụ:

```
class Simple{
private:
    int m_nID;

public:
    Simple(int nID) : m_nID{ nID } {
        cout << "Constructing Simple " << nID << '\n';
    }

    ~Simple() {
        cout << "Destructing Simple" << m_nID << '\n';
    }

    int getID() { return m_nID; }
};

int main(){
    Simple simple{ 1 };
    cout << simple.getID() << '\n';

    // Allocate a Simple dynamically
    Simple *pSimple{ new Simple{ 2 } };
    cout << pSimple->getID() << '\n';

    delete pSimple;

    return 0;
}
```

5. Con trỏ và mảng đối tượng

Con trỏ 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>;

5. Con trỏ và mảng đối tượng

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;

Khi đó, myCar là một con trỏ đối tượng có kiểu lớp Car.

- Cấp phát bộ nhớ cho con trỏ đối tượng

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>])

5. Con trỏ và mảng đối tượng

Khai báo mảng tĩnh các đối tượng

- Mảng các đối tượng được khai báo theo cú pháp:

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

- Ví dụ:

Car cars[10];

Khai báo mảng động với con trỏ

- 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>];

- Ví dụ:

Car *cars = new Car[n];

QUESTIONS & ANSWERS