Môn PPLTHĐT

Hướng dẫn thực hành tuần 7

Mục đích

Tìm hiểu về kế thừa và các vấn đề liên quan.

Nội dung

- Sơ lược về kế thừa.
- Hàm dựng và hàm hủy trong kế thừa.
- Định nghĩa lại phương thức.
- Quan hệ IS-A và HAS-A.
- Đa kế thừa trong C++.
- Vấn đề hình thoi và kế thừa ảo.

Yêu cầu

Nắm vững những nội dung trình bày trong các bài hướng dẫn thực hành từ tuần 1 đến tuần 6.

1. Sơ lược về kế thừa.

Khái niệm

Kế thừa *(inheritance)* là một tính chất quan trọng trong lập trình hướng đối tượng. Đó là khả năng định nghĩa một lớp đối tượng dựa trên một hoặc nhiều lớp đối tượng khác đã được định nghĩa trước đó. Lớp đối tượng được dùng để định nghĩa gọi là lớp cơ sở *(base class)*, lớp đối tượng được định nghĩa dựa trên những lớp đối tượng khác gọi là lớp kế thừa *(derive class)*.

Lớp kế thừa thừa hưởng đầy đủ những tính chất được định nghĩa trong lớp cơ sở. Việc kế thừa có thể được thực hiện thành nhiều cấp tạo nên cây kế thừa.

Trong C++, kế thừa được khai báo theo cấu trúc sau:
class <*Tên lớp kế thừa*> **:** <*Loại kế thừa*> <*Tên lớp cơ sở*>
{
private:
    <*Khai báo thành phần riêng*>

public:
    <*Khai báo thành phần công cộng*>
};

Ví dụ chúng ta xây dựng lớp Car dựa trên lớp Vehicle như sau:

```
class Vehicle
{
private:
    float    m_fWeight;

public:
    float GetWeight()
    {
        return this->m_fWeight;
    }

    void SetWeight(float fWeight)
    {
        this->m_fWeight = fWeight;
    }
};

class Car : public Vehicle
{
private:
    int      m_iNumberOfSeats;
```

```
public:
	int GetNumberOfSeats()
	{
		return this->m_iNumberOfSeats;
	}

	void SetNumberOfSeats(int iNumberOfSeats)
	{
		this->m_iNumberOfSeats = iNumberOfSeats;
	}
};
```

Do lớp kế thừa được định nghĩa dựa trên lớp cơ sở, nên lớp Car thừa hưởng thuộc tính m_fWeight và 2 phương thức GetWeight và SetWeight từ lớp Vehicle. Chúng ta chỉ việc định nghĩa thêm các thuộc tính và phương thức khác cho lớp Car.

Tầm vực

Tầm vực của các thành phần trong lớp đối tượng được cho bởi bảng sau:

Tầm vực	Sử dụng bên trong lớp	Sử dụng từ lớp kế thừa	Sử dụng bên ngoài lớp
Public	X	X	X
Protected	X	X	
Private	X		

Lớp kế thừa không thể truy xuất được các thành phần có tầm vực là private trong lớp cơ sở. Trong ví dụ trên, do thuộc tính m_fWeight của lớp Vehicle có tầm vực là private nên trong lớp Car, chúng ta không thể truy xuất được thuộc tính này.

Trong lớp kế thừa, để truy xuất đến thành phần của lớp cơ sở, ta dùng toán tử "::", ví dụ:

```
class A
{
public:
	void funcA()
	{
		cout << "funcA of class A.";
	}
};

class B : public A
{
public:
	void funcA()
	{
```

```
            cout << "funcA of class B.";
        }

        void funcB()
        {
            funcA();        // Goi phuong thuc funcA cua lop B.
            A::funcA();     // Goi phuong thuc funcA cua lop A.
        }
};
```

Phân loại kế thừa

Có 3 hình thức kế thừa trong C++: public, protected và private.
Tầm vực của các thành phần trong lớp đối tượng qua các hình thức kế thừa được cho bởi bảng sau:

Tầm vực	Kế thừa public	Kế thừa protected	Kế thừa private
public	public	protected	private
protected	protected	protected	private
private	Không thể truy xuất	Không thể truy xuất	Không thể truy xuất

Ví dụ:

```
class A
{
private:
    int     m_iPrivate;

protected:
    int     m_iProtected;

public:
    int     m_iPublic;
};

// Ke thua public
// - m_iPrivate khong the truy xuat.
// - m_iProtected giu nguyen tam vuc protected.
// - m_iPublic giu nguyen tam vuc public.
class B : public A {};

// Ke thua protected
// - m_iPrivate khong the truy xuat.
// - m_iProtected giu nguyen tam vuc protected.
// - m_iPublic chuyen thanh tam vuc protected.
class C : protected A {};
```

```
// Ke thua private
// - m_iPrivate khong the truy xuat.
// - m_iProtected chuyen thanh tam vuc private.
// - m_iPublic chuyen thanh tam vuc private.
class D : private A{};
```

2. Hàm dựng và hàm hủy trong kế thừa.

Hàm dựng trong kế thừa

Khi một đối tượng thuộc lớp kế thừa được tạo lập, hàm dựng của lớp cơ sở được gọi thực hiện trước, sau đó mới đến hàm dựng của lớp kế thừa. Trong hàm dựng của lớp kế thừa, chúng ta có thể chỉ ra sẽ gọi hàm dựng nào của lớp cơ sở. Trường hợp không chỉ ra, hàm dựng mặc định của lớp cơ sở sẽ được gọi.

Xét ví dụ sau:

```
class A
{
public:
	A()
	{
		cout << "Default constructor of class A." << endl;
	}
};

class B : public A
{
public:
	B(int iVar)
	{
		cout << "Constructor(int) of class B." << endl;
	}
};

class C: public B
{
public:
	C(int iVar) : B(iVar)
	{
		cout << "Constructor(int) of class C." << endl;
	}
};

void main()
{
	C	obj(5);
}
```

Ví dụ trên cho thấy thứ tự gọi thực hiện của các hàm dựng trong cây kế thừa. Ban đầu hàm dựng mặc định của lớp A được gọi trước, kế đến là hàm dựng 1 tham số của lớp B, sau cùng là hàm dựng 1 tham số của lớp C.

Hàm hủy trong kế thừa

Khi một đối tượng thuộc lớp kế thừa bị hủy, hàm hủy của lớp kế thừa được gọi thực hiện trước, sau đó mới đến hàm hủy của lớp cơ sở.

Xét ví dụ sau:

```
class A
{
public:
	virtual ~A()
	{
		cout << "Destructor of class A." << endl;
	}
};

class B : public A
{
public:
	virtual ~B()
	{
		cout << "Destructor of class B." << endl;
	}
};

class C : public B
{
public:
	virtual ~C()
	{
		cout << "Destructor of class C." << endl;
	}
}

void main()
{
	C	obj;
}
```

Ví dụ trên cho ta thấy thứ tự gọi thực hiện của các hàm hủy trong cây kế thừa. Ban đầu hàm hủy của lớp C được gọi trước, kế đến là hàm hủy của lớp B, sau cùng là hàm hủy của lớp A.

3. Định nghĩa lại phương thức

Trong một số trường hợp, chúng ta cần định nghĩa lại các phương thức của lớp cơ sở trong lớp kế thừa (chỉ những phương thức có tầm vực là public và protected). Việc này được thực hiện bằng cách khai báo và cài đặt lại các phương thức này trong lớp kế thừa. Phương thức được định nghĩa lại còn được gọi là hàm nạp chồng *(overriden function)*.

Ví dụ:

```
#include <iostream>
#include <vector>

using namespace std;

class Animal
{
public:
	vector<Animal *> GiveBirth() { // Codes for GiveBirth... }
	int GiveMilk() { // Codes for GiveMilk... }
	void Talk()
	{
		cout << "Don't know how to talk!";
	}
};

class Cat : Animal
{
public:
	void Talk()
	{
		cout << "meo meo!";
	}
};

class Dog : Animal
{
public:
	void Talk()
	{
		cout << "gau gau!";
	}
};
```

Trong ví dụ trên, lớp Cat và Dog kế thừa từ lớp Animal để tái sử dụng lại các phương thức GiveBirth() và GiveMilk(). Nhưng bên cạnh đó, chúng đã định nghĩa lại phương thức Talk() của lớp Animal cho phù hợp.

4. Quan hệ "IS-A" và "HAS-A"

Quan hệ IS-A

Do lớp kế thừa được định nghĩa dựa trên lớp cơ sở, các đối tượng của lớp kế thừa được xem như cùng loại với các đối tượng của lớp cơ sở. Vì vậy, chúng ta **không thể tự tiện thực hiện việc kế thừa** trong mọi trường hợp muốn tái sử dụng thuộc tính và phương thức của một lớp.

Việc kế thừa giữa 2 lớp **chỉ nên được thực hiện khi và chỉ khi giữa chúng có quan hệ "IS-A"**, có nghĩa là lớp này là một trường hợp đặc biệt của lớp kia.

Ví dụ:
Car là một trường hợp đặc biệt của Vehicle, do đó Car có thể kế thừa từ Vehicle.
Cat và Dog là những trường hợp đặc biệt của Animal, do đó Cat và Dog có thể kế thừa từ Animal.

Quan hệ HAS-A

Hai lớp được gọi là có quan hệ "HAS-A" khi thành phần thuộc tính của lớp này có chứa đối tượng của lớp kia. Quan hệ "HAS-A" còn được gọi là quan hệ "bao hàm" hoặc quan hệ "bộ phận – toàn thể".

Ví dụ:
Wheel chứa trong Car, do đó Car và Wheel có quan hệ "HAS-A".
Page chứa trong Book, do đó Book và Page có quan hệ "HAS-A".

Chú ý:

- Trong những trường hợp ngữ nghĩa không đóng vai trò quan trọng, quan hệ "HAS-A" có thể được sử dụng để thay thế cho quan hệ "IS-A".
- Quan hệ "IS-A" không thể được sử dụng để thay thế cho quan hệ "HAS-A".

5. Đa kế thừa trong C++

- Đa kế thừa là gì? Mục đích của đa kế thừa?
  (Từ khóa: multiple inheritance, multiple inheritance in C++)

- Cách thức khai báo và sử dụng đa kế thừa trong C++ thế nào? Tìm một cài đặt minh họa về đa kế thừa trong C++.
  (Từ khóa: multiple inheritance C++ sample code)

6. Vấn đề hình thoi và kế thừa ảo

- Vấn đề hình thoi *(diamond problem)* trong lập trình hướng đối tượng là gì? Hãy nêu nguyên nhân và đề xuất cách thức giải quyết.

(Từ khóa: diamond problem)

- Thế nào là kế thừa ảo? Nêu cách thức khai báo và sử dụng kế thừa ảo trong C++.
  (Từ khóa: virtual inheritance, virtual inheritance in C++)