Môn PPLTHĐT

Hướng dẫn thực hành tuần 8

Mục đích

Giới thiệu một số khái niệm về đa hình trong lập trình hướng đối tượng, xem xét vấn đề virtual destructor.

Nội dung

- Hàm ảo và liên kết động.
- Hàm thuần ảo và lớp trừu tượng.
- Virtual destructor.

Yêu cầu

Nắm vững những nội dung được trình bày trong các bài hướng dẫn thực hành từ tuần 1 đến tuần 7.

1. Hàm ảo và liên kết động

Một tính chất quan trọng trong kế thừa là lớp kế thừa có thể đóng vai trò của lớp cơ sở. Điều này có nghĩa là chúng ta có thể truy xuất lớp kế thừa thông qua con trỏ của lớp cơ sở. Để hiểu rõ tính chất này chúng ta xét ví dụ sau:

```
#include "iostream.h"

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

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

void main()
{
	Cat		obj;
	Animal		*p = &obj;

	obj.Talk();
	p->Talk();
}
```

Trong ví dụ trên, obj là đối tượng thuộc lớp Cat, p là con trỏ thuộc lớp Animal. Do lớp Cat kế thừa từ lớp Animal nên đối tượng của lớp Cat có thể đóng vai trò đối tượng của lớp Animal. Con trỏ p trỏ đến đối tượng obj, lúc này thông qua p, chúng ta có thể cư xử với đối tượng obj như là một đối tượng của lớp Animal. Vì vậy, p->Talk() sẽ gọi thực hiện phương thức Talk() của lớp Animal.

Bây giờ chúng ta sửa lại phương thức Talk() của lớp Animal như sau:

```
virtual void Talk()
{
	cout << "Don't know how to talk!" << endl;
}
```

Lúc này p->Talk() sẽ gọi thực hiện phương thức Talk() của lớp Cat, chính là lớp của đối tượng mà p đang trỏ đến. Hàm Talk() của lớp Animal khi đó được gọi là **hàm ảo**.

Chỉnh sửa lại hàm main() như sau:

```
void main()
{
	Animal		a;
	Cat		c;
	Animal		*p;

	p = &a;
	p->Talk();
	p = &c;
	p->Talk();
}
```

Xem xét các dòng p->Talk(), chúng ta thấy rằng việc p->Talk() sẽ gọi thực hiện hàm nào tùy thuộc vào p đang trỏ đến đối tượng nào. Điều này gọi là **liên kết động** trong lập trình hướng đối tượng.

2. Hàm thuần ảo và lớp trừu tượng

Hàm thuần ảo

Hàm thuần ảo *(pure virtual function)* là hàm ảo nhưng chỉ có phần khai báo mà không có phần cài đặt. Hàm thuần ảo được dùng trong trường hợp việc cài đặt của nó trong lớp cơ sở là không quan trọng, sự xuất hiện của nó chỉ có ý nghĩa đối với những lớp kế thừa. Chính những lớp kế thừa sẽ phải viết cài đặt cho những hàm này.

Trong C++, để khai báo hàm thuần ảo, chúng ta thêm " = 0" vào phía sau hàm ảo. Xét ví dụ sử dụng hàm thuần ảo:

```
#include "iostream.h"

class Animal
{
public:
	vector<Animal *> GiveBirth();
	int GiveMilk();
	virtual void Talk() = 0;		// Ham thuan ao, viet cai dat o lop ke thua.
};

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

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

void main()
{
	Cat	c;
	Dog	d;

	c.Talk();
	d.Talk();
```

```
}
```

Trong ví dụ trên, hàm Talk() của lớp Animal là hàm thuần ảo, chúng ta không cần viết cài đặt cho nó. Các lớp Cat và Dog kế thừa từ Animal phải viết phần cài đặt này.

Lớp trừu tượng

Lớp trừu tượng *(abstract class)* là lớp đối tượng có chứa ít nhất một hàm thuần ảo. Do các hàm thuần ảo chỉ có phần khai báo mà không có phần cài đặt nên chúng ta không thể tạo đối tượng từ lớp trừu tượng. Như vậy lớp trừu tượng chỉ được dùng để những lớp khác kế thừa.

Xét những ví dụ sau để hiểu rõ hơn về lớp trừu tượng:

```
Animal      a;                  // Loi, khong the tao doi tuong tu lop truu tuong.
Animal      *p = new Animal;    // Loi, khong the tao doi tuong tu lop truu tuong.
Animal      *q = new Cat;       // Duoc phep.

Cat         obj;
Animal      *r = &obj;          // Duoc phep.
```

3. Virtual destructor

Xét ví dụ sau:

```
class A
{
private:
	int	*m_pDataOfA;

public:
	A()
	{
		m_pDataOfA = new int[1000];
	}

	~A()
	{
		cout << "Destructor of class A.";
		delete []m_pDataOfA;
	}
};

class B: public A
{
private:
	int	*m_pDataOfB;

public:
	B()
	{
		m_pDataOfB = new int[1000];
	}

	~B()
	{
		cout << "Destructor of class B.";
		delete []m_pDataOfB;
	}
};

void main()
{
	A	*p = new B;
	delete p;
}
```

Trong đoạn chương trình trên, một đối tượng thuộc lớp B được khởi tạo và con trỏ p thuộc lớp A giữ địa chỉ của đối tượng này. Khi thực hiện delete p, hàm hủy của lớp A được gọi và vùng nhớ cấp phát cho m_pDataOfA được giải phóng. Còn **hàm hủy của lớp B không được gọi** nên vùng nhớ cấp phát cho m_pDataOfB vẫn không được giải phóng!!! Nguyên nhân là do chúng ta đang truy xuất đến đối tượng thuộc lớp B thông qua con trỏ thuộc lớp A. Và điều này làm cho đối tượng thuộc lớp B cư xử giống như nó đang là đối tượng thuộc lớp A (giống trường hợp hàm Talk() của lớp Animal ở phần 1).

Để giải quyết vấn đề này, hàm hủy của lớp A cần phải là hàm ảo. Lúc này delete p sẽ gọi hàm hủy của B, chính là lớp của đối tượng mà p đang trỏ đến. ***Đây chính là lý do hàm hủy của lớp đối tượng luôn phải được khai báo là hàm ảo.***