Môn PPLTHĐT

Hướng dẫn thực hành tuần 3

Mục đích

Tìm hiểu 3 vấn đề về con trỏ trong lập trình hướng đối tượng với hàm hủy, hàm dựng sao chép và toán tử gán.

Nội dung

- Hàm hủy và vấn đề con trỏ.
- Hàm dựng sao chép và vấn đề con trỏ.
- Toán tử gán và vấn đề con trỏ.

Yêu cầu

Nắm vững những nội dung trình bày trong bài hướng dẫn thực hành số 2.

1. Hàm hủy và vấn đề con trỏ

Xét ví dụ xây dựng lớp mảng các số nguyên IntegerArray.

Bước 1: vào VS, tạo project dạng Console Application (Visual C++).

Bước 2: thêm vào project file IntegerArray.h và viết khai báo lớp IntegerArray như sau:

```
class IntegerArray
{
private:
	int	*m_pElement;
	int	m_iLength;

public:
	IntegerArray(int iLength);

	int GetLength();
	int & ElementAt(int iIndex);
	int & operator [](int iIndex);
};
```

Bước 3: thêm vào project file IntegerArray.cpp và viết cài đặt lớp IntegerArray như sau:

```
#include "iostream.h"
#include "IntegerArray.h"

IntegerArray::IntegerArray(int iLength)
{
	if (iLength < 0)
	{
		cout << "Loi: chieu dai mang la so am.";
		return;
	}

	m_iLength = iLength;
	m_pElement = new int[m_iLength];
}

int IntegerArray::GetLength()
{
	return m_iLength;
}

int & IntegerArray::ElementAt(int iIndex)
{
	if (iIndex < 0 || iIndex >= m_iLength)
```

```
                cout << "Loi: truy xuat phan tu ngoai pham vi mang.";

        return m_pElement[iIndex];
}

int & IntegerArray::operator [](int iIndex)
{
        return ElementAt(iIndex);
}
```

Bước 4: thêm vào project file main.cpp và viết đoạn chương trình sử dụng lớp IntegerArray vừa tạo như sau:

```
#include "IntegerArray.h"

void main()
{
        IntegerArray    a(3);

        a[0] = 0;
        a[1] = 1;
        a[2] = 2;

        a.ElementAt(1) = 3;
        a.ElementAt(2) = 4;

        for (int i = 0; i < a.GetLength(); i++)
                cout << a[i] << endl;
}
```

Bước 5: biên dịch và chạy thử chương trình.

Trong đoạn chương trình trên, sau khi kết thúc hàm main(), đối tượng a bị hủy, vùng nhớ mà m_pElement và m_iLength chiếm chỗ được giải phóng. Nhưng vùng nhớ cấp phát cho m_pElement lại không được giải phóng. Điều này gây ra rò rỉ bộ nhớ *(memory leak)*, đặc biệt nghiêm trọng trong trường hợp vùng nhớ cấp phát cho m_pElement lớn. Để giải quyết vấn đề này, chúng ta phải viết lệnh giải phóng vùng nhớ cấp phát cho m_pElement trong hàm hủy của lớp IntegerArray.

Trong file IntegerArray.h thêm vào khai báo hàm hủy cho lớp IntegerArray như sau:

```
virtual ~IntegerArray();
```

Trong file IntegerArray.cpp viết cài đặt cho hàm hủy như sau:

```
IntegerArray::~IntegerArray()
{
        if (m_pElement != NULL)
                delete []m_pElement;
```

}

Như vậy, khi lớp đối tượng có thuộc tính kiểu con trỏ và cấp phát vùng nhớ thì phải xây dựng hàm hủy cho lớp đối tượng đó để giải phóng vùng nhớ đã cấp phát.

2. Hàm dựng sao chép và vấn đề con trỏ

Khái niệm

Hàm dựng sao chép *(copy constructor)* là hàm dựng khởi tạo đối tượng dựa trên một đối tượng có cùng kiểu.

Các tính chất của hàm dựng sao chép:

- Hàm dựng sao chép có đầy đủ tính chất của một hàm dựng thông thường.
- Thường được dùng để tạo bản sao của đối tượng.
- Trong C++, hàm dựng sao chép của lớp A được khai báo như sau:

```
class A
{
public:
	A(const A &obj);
};
```

Bất kỳ lớp đối tượng nào cũng có hàm dựng sao chép. Trong trường hợp chúng ta không khai báo hàm dựng sao chép cho lớp đối tượng, hàm dựng sao chép mặc định *(default copy constructor)* sẽ được tự động được thêm vào.

Ví dụ

Để hiểu rõ hơn về hàm dựng sao chép, chúng ta chỉnh sửa lại hàm main() của ví dụ lớp IntegerArray bên trên như sau:

```
void main()
{
	IntegerArray	a(3);

	a[0] = 0;
	a[1] = 1;
	a[2] = 2;

	a.ElementAt(1) = 3;
	a.ElementAt(2) = 4;

	IntegerArray	b(a);	// Tao doi tuong b la ban sao cua doi tuong a.

	for (int i = 0; i < b.GetLength(); i++)
		cout << b[i] << endl;
}
```

Biên dịch và chạy thử chương trình chúng ta nhận được thông báo lỗi ngay tại hàm hủy của lớp IntegerArray. **Chuyện gì đã xảy ra???**

Phân tích chương trình chúng ta thấy rằng lớp IntegerArray không khai báo hàm dựng sao chép, do đó hàm dựng sao chép mặc định sẽ được tự động thêm vào. Hàm dựng sao chép mặc định tạo bản sao từ đối tượng bằng cách sao chép từng thuộc tính của đối tượng đó, nhưng lại không sao chép vùng nhớ cấp phát cho những thuộc tính kiểu con trỏ. Như vậy m_pElement của đối tượng b và m_pElement của đối tượng a sẽ sử dụng chung một vùng nhớ. Khi kết thúc hàm main(), đối tượng b bị hủy trước, hàm hủy của b được gọi để hủy vùng nhớ cấp phát cho m_pElement của b. Kế đó đối tượng a bị hủy, hàm hủy của a cũng được gọi để hủy vùng nhớ cấp phát cho m_pElement của a. Nhưng bấy giờ vùng nhớ này đã bị hủy trước đó. Chính điều này gây ra lỗi chương trình.

Hơn nữa việc 2 mảng sử dụng chung một vùng nhớ để lưu các phần tử rõ ràng là sẽ gây ra những sai lệch về mặt ngữ nghĩa của chương trình.

Để giải quyết vấn đề này, chúng ta phải xây dựng hàm tạo sao chép cho lớp IntegerArray. Trong file IntegerArray.h, thêm vào khai báo hàm dựng sao chép cho lớp IntegerArray như sau:

```
IntegerArray(const IntegerArray &obj);
```

Trong file IntegerArray.cpp viết cài đặt cho hàm dựng sao chép

```
IntegerArray::IntegerArray(const IntegerArray &obj)
{
	m_iLength = obj.m_iLength;
	m_pElement = new int[m_iLength];

	// Sao chep vung nho da cap phat cho m_pElement cua obj.
	for (int i = 0; i < m_iLength; i++)
		m_pElement[i] = obj.m_pElement[i];
}
```

Như vậy, khi lớp đối tượng có thuộc tính kiểu con trỏ và cấp phát vùng nhớ thì phải xây dựng hàm dựng sao chép cho lớp đối tượng đó để sao chép vùng nhớ đã cấp phát.

3. Toán tử gán

Vẫn sử dụng lớp IntegerArray xây dựng ở trên, chúng ta chỉnh sửa lại hàm main() như sau:

```
void main()
{
	IntegerArray	a(3);

	a[0] = 0;
	a[1] = 1;
	a[2] = 2;

	a.ElementAt(1) = 3;
	a.ElementAt(2) = 4;

	IntegerArray	b(3);

	b = a;	// Tao ban sao tu doi tuong a bang toan tu gan.

	for (int i = 0; i < b.GetLength(); i++)
		cout << b[i] << endl;
}
```

Biên dịch và chạy thử chương trình chúng ta nhận được thông báo lỗi ngay tại hàm hủy của lớp IntegerArray. **Chuyện gì đã xảy ra???**

Phân tích toán tử gán chúng ta thấy rằng khi gán đối tượng x bằng đối tượng y (x, y cùng kiểu) thì tất cả các thuộc tính của x sẽ được gán giá trị bằng các thuộc tính tương ứng của y. Điều này có nghĩa là trong ví dụ trên m_pElement của đối tượng b sẽ được gán giá trị bằng m_pElement của đối tượng a. Như vậy m_pElement của b và m_pElement của a sẽ cùng trỏ đến một vùng nhớ. Đây chính là vấn đề chúng ta đã gặp phải khi đề cập đến hàm tạo sao chép ở trên.

Để giải quyết vấn đề này, chúng ta phải định nghĩa lại toán tử gán cho lớp IntegerArray. Trong file IntegerArray.h, thêm vào khai báo toán tử gán cho lớp IntegerArray như sau:

```
IntegerArray & operator =(const IntegerArray &obj);
```

Trong file IntegerArray.cpp viết cài đặt cho hàm dựng sao chép

```
IntegerArray & IntegerArray::operator =(const IntegerArray &obj)
{
	m_iLength = obj.m_iLength;
	m_pElement = new int[m_iLength];

	// Sao chep vung nho da cap phat cho m_pElement cua obj.
	for (int i = 0; i < m_iLength; i++)
```

```
            m_pElement[i] = obj.m_pElement[i];

    return *this;
}
```

Như vậy, khi lớp đối tượng có thuộc tính kiểu con trỏ và cấp phát vùng nhớ thì phải định nghĩa lại toán tử gán cho lớp đối tượng đó để sao chép vùng nhớ đã cấp phát.