Bài thực hành số 4

Mục tiêu:

- Viết chương trình có sử dụng cơ chế tương ứng bội (polymorphism).
- Làm rõ các khái niệm: hàm ảo, hàm thuần ảo và hàm hủy ảo.

1. Cơ chế tương ứng bội (Polymorphism).

Cơ chế tương ứng bội (tính đa hình) trong C++ là cơ chế cho phép hành vi đối tượng có thể có nhiều thể hiện khác nhau tùy thuộc vào lớp thực chất mà đối tượng đó thuộc về. Khả năng cho phép một chương trình sau khi đã biên dịch có thể có nhiều diễn biến xảy ra là một trong những thể hiện của tính đa hình – tính muôn màu muôn vẻ – của chương trình hướng đối tượng.

Để thực hiện được tính đa hình, C++ có cơ chế kết nối động (dynamic binding) bằng cách sử dụng hàm ảo (virtual functions) thay cho cơ chế kết nối tĩnh (static binding) ngay khi chương trình biên dịch.

2. Hàm ảo (Virtual functions)

Hàm ảo là hàm phải được mô tả trong lớp cơ sở, trước hàm ảo phải có từ khoá virtual.

Ví dụ 1. Kết quả chương trình là gì?

```
// pure virtual members can be called
// from the abstract base class
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area() =0;
    void printarea()
      { cout << this->area() << '\n'; }
};

class Rectangle: public Polygon {
  public:
    int area (void)
      { return (width * height); }
};

class Triangle: public Polygon {
  public:
    int area (void)
      { return (width * height / 2); }
};

int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

Ví dụ 2. Kết quả chương trình là gì?

```
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class Base                          //base class
   {
   public:
      virtual void show() = 0;     //pure virtual function
   };
////////////////////////////////////////////////////////////////
class Derv1 : public Base           //derived class 1
   {
   public:
      void show()
         { cout << "Derv1\n"; }
   };
////////////////////////////////////////////////////////////////
class Derv2 : public Base           //derived class 2
   {
   public:
      void show()
         { cout << "Derv2\n"; }
   };
////////////////////////////////////////////////////////////////
int main()
   {
// Base bad;              //can't make object from abstract class
   Base* arr[2];          //array of pointers to base class
   Derv1 dv1;             //object of derived class 1

   Derv2 dv2;             //object of derived class 2

   arr[0] = &dv1;         //put address of dv1 in array
   arr[1] = &dv2;         //put address of dv2 in array

   arr[0]->show();        //execute show() in both objects
   arr[1]->show();
   return 0;
   }
```

Ví dụ 3. Kết quả chương trình là gì?

```
#include <iostream >
#include <string.h>
using namespace std;
class person {
      protected:
            char *name;
      public:
            person(char *n) { name = n;}
            virtual void print()        // hàm ảo
            {
                  cout << "Toi ten la "<<name;
            }
};
class foreigner : public person {
      public:
            foreigner(char *n): person(n) { }
            void print()
            {
                  cout <<"My name is " <<name;
            }
};

class alien : public person
{
      public:
             alien(char *n): person(n){ }
             void print()
            {
                  cout <<"&*^*&%& **^@<<name<<endln;
                  cout << "Sorry, there's a communication problem";
             }
};
int main(){
      person *p1, *p2, *p3;
      p1 = new person("Jack");
      p2 = new foreigner("Maria");
      cout <<"Introducing a Person:";
      p1->print();
      cout <<"Introducing a Foreigner:";
      p2->print();
      p3 = new alien("Martian");
      cout << "Introducing an Alien:";
      p3 ->print();
}
```

Kết quả sau khi chạy chương trình:

```
Introducing a Person  : Toi ten la Jack
Introducing a Foreiger: My name is Maria

Introducing an Alien  : &*^*&%& **^@ Martian
Sorry there's a communication problem
```

Trong ví dụ trên ta thấy cả hai lớp dẫn xuất đều có hàm print( ) như trong lớp cơ sở của chúng, hàm print trong lớp person là một hàm ảo. Ban đầu cả hai con trỏ p1 và p2, p3 đều có kiểu person nhưng sau đó p1 lại thực sự trỏ đến một đối tượng kiểu person trong khi p2 lại trỏ đến một đối tượng kiểu foreigner và p3 trỏ đến đối tượng kiểu alien. Khi thực hiện p2 không gọi hàm print của lớp person mà gọi hàm print của lớp foreigner, và p3 gọi hàm print của lớp alien.

Đây chính là cơ chế đa hình mà ta đã nói ở trên.

3. Hàm thuần ảo

Hàm thuần ảo là hàm ảo được mô tả bằng cách gán giá trị 0 cho hàm. Một lớp được gọi là lớp trừu tượng nếu trong lớp tồn tại một hàm ảo và không thể dùng lớp này để định nghĩa một đối tượng.

Xét chương trình sau:

```
#include <iostream>
using namespace std;
class Format {
      public:
            void display_form();
            virtual void header() {
                  cout << "This is a header\n";
            }
            virtual void body() = 0; // hàm thuần ảo
            virtual void footer() {
                  cout << "This is a footer\n\n";
            }
};
void Format :: display_form() {
      header();
      for( int index = 0; index <3 ; index ++)
      {
            body();
      }
      footer();
}

class MyForm : public Format
{
public:
      void body()
      {
          cout << "This is the new body of text\n";
      }
      void footer()
      {
          cout << This is the new footer\n";
      }

};

int main()
{
      Format* form = new MyForm;
      form->display_form();
}
```

Kết quả sau khi chạy chương trình:

```
This is a header
This is the new body of text
This is the new body of text
This is the new body of text
This is the new footer
```

Chú ý: trong trường hợp này lớp MyForm không có hàm header nên không thực hiện overriding được, do vậy trong display_form sẽ sử dụng hàm header của lớp Format. Như vậy nếu hàm header không phải là hàm ảo thì kết quả cũng không có gì khác.

4. Hàm hủy ảo

Hàm hủy là hàm được gọi một cách tự động để thực hiện một số công việc nào đó mà người lập trình cần khi đối tượng bị hủy , chẳng hạn hàm hủy dùng để giải phóng bộ nhớ một cách tự động khi đối tượng bị hủy. Tuy nhiên bình thường thì hàm hủy của lớp dẫn xuất sẽ không được gọi mà chỉ có hàm hủy của lớp cơ sở được gọi , nên trong một số trường hợp ta không thể dùng hàm ảo trong một lớp dẫn xuất để thực hiện các thao tác mà ta cần khi một làm khi một đối tượng của lớp mày bị hủy. Vấn đề này được giải quyết bằng cách dùng hàm hủy ảo.

Hàm hủy của lớp cơ sở sẽ là một hàm ảo, các lớp dẫn xuất sẽ có hàm hủy riêng và khi đối tượng bị hủy, hàm hủy của lớp cơ sở và lớp dẫn xuất đều sẽ được gọi.

Xét ví dụ sau:

```
#include <iostream>
using namespace std;
class Base {
public:
       ~Base()
      {
            cout<<"~Base"<<endl;
       }
};
class Derived:public Base {
public:
      ~Derived(){ cout<<"~Derived"<<endl; }
};
int main()
{
      Base *B;
      B = new Derived;
      delete B;
}
```

Kết quả khi thực hiện:

```
~Base
```

Như vậy ta thấy chỉ có hàm hủy của lớp cơ sở được gọi , còn hàm hủy của lớp dẫn xuất không được gọi vì hàm huỷ của lớp cơ sở không phải là hàm ảo, B là con trỏ kiểu Base thì khi thực hiện phương thức huỷ sẽ gọi hàm huỷ của Base.

Bây giờ ta sửa lại chương trình trên như sau:

```
#include <iostream>
using namespace std;
class Base
{
public:
	virtual ~Base()
	{
		cout<<"~Base"<<endl;
	}
};

class Derived: public Base
{
public:
	virtual ~Derived()
	{
		cout<<"~Derived"<<endl;
	}
};

int main()
{
	Base *B;
	B = new Derived;
	delete B;
}
```

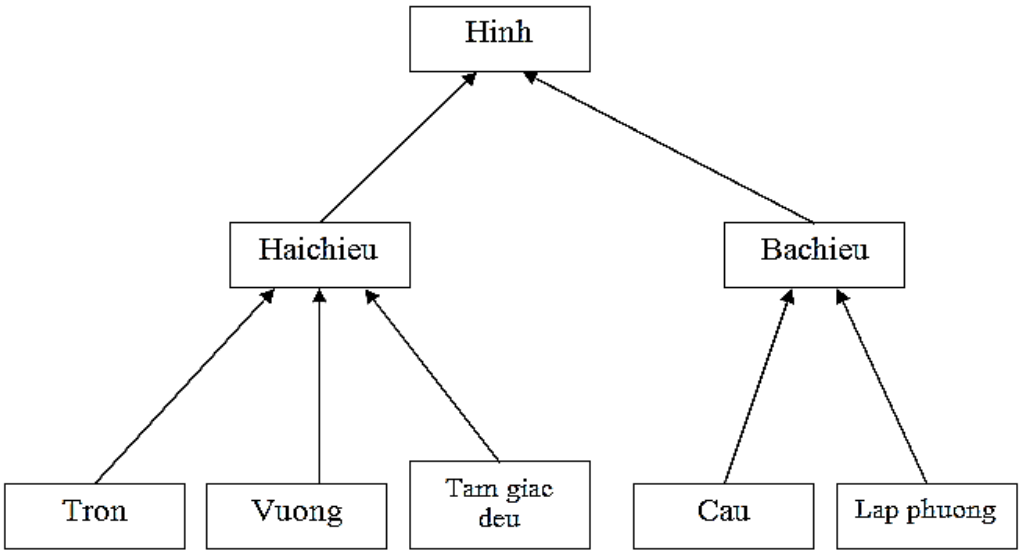
Khi đó kết quả của chương trình sẽ là:

```
~Derived
~Base
```

Bài tập làm thêm

TH4.1.

Xây dựng các lớp theo cây kế thừa sau:



Yêu cầu:

- Tất cả các loại hình đều có chung phương thức `ten()` để xác định tên hình là gì và phương thức `in()` để thể hiện tên hình
- Các hình hai chiều đều có phương thức `dt()` để tính diện tích
- Các hình ba chiều đều có phương thức `tt()` để tính thể tích