**Kỹ Thuật Lập Trình**

**Học kỳ 2 – Năm học 2017-2018**

**Lab 9 – Lập trình hướng đối tượng**

# MỤC TIÊU CỦA BÀI THỰC HÀNH

* Hiểu rõ các khái niệm căn bản của hướng đối tượng.
* Có thể tạo một lớp (class) và dùng lớp đó bằng cách tạo đối tượng từ lớp và gọi
* phương thức của đối tượng.
* Biết cách thiết kế lớp đơn giản để giải quyết bài toán.
* Hiểu và sử dụng tính thừa kế
* Hiểu được sơ đồ lớp cơ bản
* Biết về tính đa hình (polymorphism)

# BÀI TẬP BẮT BUỘC

**Câu 1:** Cho chương trình sau đây, nằm giữa “BEGIN PROGRAM” và “END PROGRAM”.

a) Chạy chương trình đó.

b) Chương trình này đó in ra những gì trên màn hình khi kết thúc thực thi. Chú ý, trên Windows, các bạn phải chạy ở chế độ dòng lệnh, mới thấy hết các dòng chương trình in ra? Hãy giải thích kết quả được in ra.

Ghi ra kết quả ở các dòng sau đây:

L1: ….

L2: …. (tự thêm tiếp các dòng khác)

c) Hàm ~A(): là hàm gì? Được gọi khi nào?

d) Nếu xoá bỏ dòng “public:”, biên dịch lại chương trình có lỗi không? Vì sao?

e) Nếu thay chữ “private” thành “protected”, biên dịch lại chương trình có lỗi không? Vì sao?

f) Nếu xoá bỏ toàn bộ hàm khởi tạo A(). Biên dịch lại chương trình có lỗi không? Vì sao?

g) Nếu xoá bỏ toàn bộ hàm khởi tạo A(const A& ob). Biên dịch lại chương trình có lỗi không? Vì sao? Nếu biên dịch thành công, chạy thì in ra những gì?

h) Nếu xoá bỏ toàn bộ hàm huỷ. Biên dịch lại chương trình có lỗi không? Vì sao? Nếu biên dịch thành công, chạy thì in ra những gì?

i) Thay vì khởi động biến m\_value trên dòng khởi tạo, như “A():m\_value(123)”, có cách nào khác? Hãy thực hiện theo cách đó, biên dịch và chạy.

----- BEGIN PROGRAM------

#include <iostream>

#include <iomanip>

using namespace std;

class A{

private:

int m\_value;

public:

A(): m\_value(123){

cout << "In A(), value = " << m\_value << endl;

}

A(const A& ob): m\_value(ob.m\_value){

cout << "In A(const A& ob), value = " << m\_value << endl;

}

A(int value): m\_value(value){

cout << "In A(int value), value = " << m\_value << endl;

}

~A(){

cout << "In ~A(), value = " << m\_value << endl;

}

};

int main(){

A ob1;

A \*ptr1 = new A(246);

{

A ob2(456);

A ob3(ob1);

}

delete ptr1;

A ob4(789);

A ob5 = 357;

return 0;

};

----- END PROGRAM---------

**Hướng dẫn:**

1) Copy chương trình, biên dịch và chạy. Lưu ý, để phải được cả những kết quả *sau khi chương trình kết thúc*.

2) Chỉnh sửa từng bước theo các yêu cầu; sau đó, biên dịch, chạy nếu được, giải thích kết quả.

**Câu 2:**

Cho chương trình sau đây, nằm giữa “BEGIN PROGRAM” và “END PROGRAM”.

a) Hãy điền vào //(1) các dòng để khởi động giá trị m\_value trong lớp X, Y, và Z với

các giá trị của thông số vx, vy, và vz tương ứng.

b) Hãy điền vào //(2) dòng lệnh để xuất 3 giá trị m\_value (có 3 giá trị có tên m\_value)

trong đối tượng “obj” ra màn hình, theo định dạng: vx, vy, vz.

----- BEGIN PROGRAM------

class X{

public:

int m\_value;

};

class Y: public X{

public:

int m\_value;

};

class Z: public Y{

public:

Z(int vx, int vy, int vz){

//(1)

}

public:

int m\_value;

};

int main(){

Z obj(1,2,3);

//(2)

return 0;

};

----- END PROGRAM---------

**Hướng dẫn:**

1) Copy chương trình, biên dịch và chạy.

2) Để truy cập các thành viên được định nghĩa lại (overriden) thì dùng toán tử phân giải tầm vực – xem slide bài giảng.

**Câu 3:**

Cho chương trình sau đây, nằm giữa “BEGIN PROGRAM” và “END PROGRAM”.

a) Biên dịch chương trình có lỗi không? Nếu không lỗi thì khi chạy, chương trình in ra gì trên màn hình?

b) Nếu thay từ khoá public trong “class Z: public Y” thành protected, biên dịch lại có lỗi không? Vì sao?

c) Nếu thay từ khoá public trong “class Y: public X” thành protected, biên dịch lại có

lỗi không? Vì sao?

d) Nếu thay từ khoá public trong “lớp X” thành protected, biên dịch lại có lỗi không? Vì sao?

----- BEGIN PROGRAM------

class X{

public:

void display(){

cout << "class X" << endl;

}

};

class Y: public X{

public:

void display(){

cout << "class Y" << endl;

}

};

class Z: public Y{

public:

void display(){

cout << "class Z" << endl;

}

};

int main(){

Z obj;

obj.X::display();

obj.Y::display();

obj.Z::display();

obj.display();

return 0;

};

----- END PROGRAM---------

**Hướng dẫn:**

1) Copy chương trình, biên dịch và chạy.

2) Thay đổi từng bước theo yêu cầu; sau đó, biên dịch và chạy nếu được. Giải thích kết quả.

**Câu 4:**

Cho chương trình sau đây, nằm giữa “BEGIN PROGRAM” và “END PROGRAM”.

a) Biên dịch, chạy, và giải thích kết quả

b) Thêm từ khoá “virtual” vào chổ /\*(1)\*/ (trước phương thức display trong lớp X). Biên dịch, chạy, và giải thích kết quả

----- BEGIN PROGRAM------

#include <iostream>

using namespace std;

class X{

public:

/\*(1)\*/ void display(){

cout << "Type of \"this\" object is: Class X" << endl;

}

};

class Y: public X{

public:

void display(){

cout << "Type of \"this\" object is: Class Y" << endl;

}

};

class Z: public Y{

public:

void display(){

cout << "Type of \"this\" object is: Class Z" << endl;

}

};

int main(){

cout << "Call method via NON-POINTER variable - CASE I:" << endl;

X x;

Y y;

Z z;

x.display();

y.display();

z.display();

/////////////////

cout << " Call method via NON-POINTER variable - CASE II:" << endl;

X x1, x2, x3;

Y y2;

Z z3;

x2 = y2; x3 = z3; //ATTENTION

x1.display();

x2.display();

x3.display();

/////////////////

cout << " Call method via POINTER variable - CASE I:" << endl;

X \*px = new X();

Y \*py = new Y();

Z \*pz = new Z();

px->display();

py->display();

pz->display();

delete px;

delete py;

delete pz;

/////////////////

cout << " Call method via POINTER variable - CASE II:" << endl;

X \*px1 = new X();

X \*px2 = new Y(); //ATTENTION

X \*px3 = new Z(); //ATTENTION

px1->display();

px2->display();

px3->display();

delete px1;

delete px2;

delete px3;

return 0;

};

----- END PROGRAM---------

**Hướng dẫn:**

1) Copy chương trình, biên dịch và chạy.

2) **Gọi phương thức không thông qua con trỏ:** phép gán đối tượng của lớp con con vào đối tượng của lớp cha (như dòng: x2 = y2; x3 = z3;) là cho phép. Vì đối tượng của lớp con chứa trong đó đối tượng có kiểu lớp cha. Nhưng ngược lại thì không được.

3) **Gọi phương thức thông qua con trỏ:** con trỏ đến đối tượng lớp cha luôn luôn gán được cho con trỏ đến đối tượng của lớp cha, như dòng: X \*px2 = new Y(); X \*px3 = new Z();. Phép gán này có ý nghĩa trong tính đa hình.

**Câu 5:**

Cho chương trình sau đây, nằm giữa “BEGIN PROGRAM” và “END PROGRAM”.

a) Thêm vào lớp Y là lớp con của lớp X. Định nghĩa lại (override) phương thức display trong X, sao cho bên ngoài lớp Y vẫn gọi được – như dòng: “y.display();” trong hàm main

b) Thêm vào lớp Z là lớp con của lớp Y. Định nghĩa lại (override) phương thức display thừa kế được từ lớp Y, sao cho bên ngoài lớp Z không thể dùng phương thức display được nữa – như dòng: “z.display();“ trong hàm main. Sinh viên phải thực hiện bằng 2 cách khác nhau:

----- BEGIN PROGRAM------

#include <iostream>

using namespace std;

class X{

public:

/\*(1)\*/ void display(){

cout << "Type of \"this\" object is: Class X" << endl;

}

};

int main(){

X x;

Y y;

Z z;

x.display(); //OK

y.display(); //OK

z.display(); //Error during compilation

return 0;

};

----- END PROGRAM---------

**Hướng dẫn:**

Câu 5.b) Cho hai cách che dấu phương thức của lớp cha:

1) Dùng thừa kế kiểu protected hay private. Ví dụ: “class Z: protected Y”

2) Vẫn dùng thừa kế kiểu public, nhưng hàm bị định nghĩa lại có tính private hay protected trong lớp Z.

**Câu 6:** Hiện thực lớp hình chữ nhật *Rectangle* theo các miêu tả về thuộc tính dữ liệu và hàm thành viên như sau:

* Có hai thuộc tính chiều dài (*length*) và chiều rộng (*width*).
* Định nghĩa hàm khởi tạo có hai thông số ứng với hai thông tin chiều dài và chiều rộng, giá trị mặc định cho hai thông số này lần lượt là 20 và 10.
* Định nghĩa thủ tục *display* để hiển thị các thuộc tính ra màn hình.
* Định nghĩa thủ tục *getInfo* để yêu cầu người dùng nhập các thông tin cho thuộc tính.
* Định nghĩa các thủ tục *set* để gán giá trị cho các thuộc tính.
* Định nghĩa các hàm *get* để truy xuất giá trị của các thuộc tính.
* Định nghĩa hàm *area* để tính diện tích của hình chữ nhật.

Hiện thực lớp hình hộp *Box* theo miêu tả sau:

* Thừa kế lớp *Rectangle* ở trên.
* Có thêm thuộc tính chiều sâu *depth*.
* Định nghĩa đè (*override*) các thủ tục *display*, *getInfo* và hàm *area* của lớp

*Rectangle*.

**Câu 7:** Một đối tượng kiểu Date lưu thông tin về ngày, tháng, và năm. Hãy tạo ra kiểu Date để người dùng (người lập trình) có thể chạy được đoạn chương trình sau. Các dòng được giải thích như sau:



* Dòng 3: Chèn phần **header** của lớp Date mà sinh viên tạo ra.
* Dòng 8: Tạo hai đối tượng có kiểu Date, giá trị ban đầu của ngày, tháng, và năm gán giá trị không hợp lệ nào đó; ví dụ: year = -1. Tốt nhất: sinh viên lấy ngày của hệ thống gán cho các thành viên của lớp Date; do đó, có thể cho phép người dùng tạo ngày chứa ngày hiện tại dễ dàng.
* Dòng 10 và 12: Lớp Date phải hỗ trợ nhập ngày bằng toán tử chèn >>. Người dùng có thể nhập ngày bằng cách gõ: **5/2/2018 và ENTER**.
* Dòng 14 và 15: Lớp Date phải hỗ trợ in ra ngày bằng toán tử chèn << cho các đối tượng Date. Ví dụ, với ngày nhập ở trên thì in ra: **05/02/2018** (*nghĩa là thêm số 0 nếu chưa đủ 2 ký tự số*).
* Dòng 16: Lớp Date phải hỗ trợ toán tử so sánh < giữa hai đối tượng Date.
* Dòng 18: Lớp Date phải hỗ trợ tạo đối tượng Date bằng cách truyền ngày, tháng và năm vào thông số của hàm khởi tạo.
* Dòng 19: Lớp Date phải hỗ trợ phương thức *toString()*. Phương thức này đổi đối tượng sang chuỗi. Cụ thể, dòng 19 sẽ in ra: **25/05/2017**.