**OOP 期末复习**

**Lecture2 类和对象、构造函数与析构函数**

**知识点**

class 类名 {

public:

公有数据成员

类名(参数列表); *// 构造函数*

类名(类名 *&*对象名); *// 拷贝构造函数，参数常用 const 修饰*

*~*类名(); *// 析构函数*

返回类型 公有成员函数(参数列表);

返回类型 显式内联成员函数(参数列表); *// 内联函数显式声明*

返回类型 隐式内联成员函数(参数列表) { *// 内联函数隐式声明*

*// 函数体*

}

protected:

保护数据成员

保护成员函数

private:

私有数据成员

私有成员函数

}**;**

类名:*:*类名(参数表) { *// 构造函数*

*// 函数体*

}

类名:*:*类名(类名 *&*对象名) { *// 拷贝构造函数*

*// 函数体*

}

类名:*:~*类名() { *// 析构函数*

*// 函数体*

}

返回类型 类名:*:*公有成员函数(参数表) {

*// 函数体*

}

*inline* 返回类型 类名:*:*显式内联成员函数(参数列表) { *// 内联函数显式声明*

*// 函数体*

}

**练习**

判断：

（ ）构造函数没有返回类型。

（ ）构造函数可以被显式调用。

（ ）如果没有显示定义构造函数，那么该类就没有构造函数。

（ ）构造函数的名字可以与类名不同。

（ ）每个类都必须有一个拷贝构造函数。

（ ）构造函数可以重载。

（ ）拷贝构造函数可以重载。

（ ）析构函数可以重载。

（ ）如果一个对象被定义在一个函数体内，当这个函数结束时，析构函数将被自动调用。

（ ）析构函数是成员函数，函数体可写在类体内，也可写在类体外。

选择：

拷贝构造函数什么时候执行：

A - 当用类的一个对象初始化该类的另一个对象的时候

B - 如果函数的形参是类的对象,调用函数时,进行形参和实参结合的时候

C - 函数的返回值是类的对象的时候

D - 以上说法都正确

写出下列代码的输出：

**提示：本题很难，请务必全面复习这一章节内容后再做。**

**相关知识点：构造函数与析构函数的调用顺序，初始化列表的初始化顺序，拷贝构造函数如何调用**

*#include* <iostream>

*using* *namespace* std;

*class* A {

*public*:

*int* a1;

*int* a2;

A();

A(*const* A *&*\_a);

A(*int* i);

~A();

};

A::A()*:* a1(10), a2(10) {

cout *<<* "Constructor 1 of A." *<<* endl;

}

A::A(*const* A *&*\_a) {

cout *<<* "Copy-constructor of A." *<<* endl;

}

A::A(*int* i)*:* a2(i*++*), a1(i*++*) {

cout *<<* "Constructor 2 of A." *<<* endl;

}

A::~A() {

cout *<<* "Destructor of A." *<<* endl;

}

*class* B {

*public*:

A b1;

*int* b2;

B();

B(*int* \_b2);

~B();

};

B::B(*int* \_b2)*:* b2(\_b2) {

cout *<<* "Constructor of B." *<<* endl;

}

B::~B() {

cout *<<* "Destructor of B." *<<* endl;

}

*int* main() {

A temp1(0);

cout *<<* "temp1.a1: " *<<* temp1.a1 *<<* endl;

cout *<<* "temp2.a2: " *<<* temp1.a2 *<<* endl;

A temp2 *=* temp1;

B temp3(15);

*return* 0;

}

**Lecture 3 类的设计**

**知识点**

*class* A {

*public*:

*int* a1;

*int* a2;

A(*int* \_a1, *int* \_a2, *int* \_a3); *// 略去实现*

*void* func() {

*this*->a1 *=* 1; *// 使用 this 指针*

}

*friend* *void* friend\_func(*const* A *&*tmp); *// 友元函数*

*private*:

*int* a3;

};

*void* func2(A tmp) { *// 传值，a1 没有改变*

tmp.a1 *=* 10;

}

*void* func3(A *\**tmp) { *// 传址，a1 改变*

tmp->a1 *=* 10;

}

*void* func4(A *&*tmp) { *// 传引用，a1 改变*

tmp.a1 *=* 10;

}

*void* friend\_func(*const* A *&*tmp) { *// 友元函数实现*

*// 函数体*

}

*int* main() {

A obj(5, 6, 7); *// 声明对象*

A *\**ptr; *// 声明指针*

ptr *=* *&*obj; *// 用指针指向对象*

cout *<<* obj.a1; *// 通过对象访问数据成员*

cout *<<* ptr->a1; *// 通过指针访问数据成员*

cout *<<* obj.a3; *// 错误，不能访问私有的数据成员*

obj.func(); *// 通过对象调用成员函数*

ptr->func(); *// 通过指针调用成员函数*

cout *<<* ptr; *// 打印一串十六进制地址*

cout *<<* obj; *// 错误，A 没有重载 <<*

*int* A::*\**a2\_ptr; *// 声明数据成员指针*

a2\_ptr *=* *&*A::a2; *// 赋值*

cout *<<* obj.\*a2\_ptr; *// 使用数据成员指针访问 a2*

cout *<<* ptr->\*a2\_ptr; *// 使用数据成员指针访问 a2*

*void* (A::*\**func\_ptr)(); *// 声明成员函数指针*

func\_ptr *=* A::func; *// 赋值*

(obj.\*func\_ptr)(); *// 使用成员函数指针*

(ptr->\*func\_ptr)(); *// 使用成员函数指针*

func2(obj); *// 传值，a1 没有改变*

func3(*&*obj); *// 传址，a1 改变*

func4(obj); *// 传引用，a1 改变*

*const* *int* temp1 *=* 5;

*int* temp2 *=* 10;

*const* *int* *\**temp3 *=* *&*temp1; *// 指向常量的指针*

*int* *\**temp4 *=* *&*temp1; *// 错误，必须要指向常量的指针才能指向常量*

*int* *\*const* temp5 *=* *&*temp2; *// 常指针*

temp3 *=* *&*temp2; *// 正确，指针可以指向其他常量或变量*

*\**temp3 *=* 0; *// 错误，不能通过指向常量的指针修改值，即使指向的并不是常量*

temp5 *=* *&*temp1; *// 错误，常指针的值不能修改*

*\**temp5 *=* 15; *// 正确，常指针指向的值不是常量，可以修改*

*return* 0;

}

**练习**

判断：

（ ）静态数据成员的初始化在类的内部完成。

（ ）类的静态数据成员必须进行初始化，对它的初始化应该在类的构造函数中进行。

（ ）无论建立多少个类的对象，都只有一个静态数据的拷贝。

（ ）静态数据成员必须用 public 修饰。

（ ）静态成员函数中有 this 指针。

（ ）普通成员函数中有 this 指针。

（ ）友元函数中有 this 指针。

（ ）友元函数就是当前类的成员函数。

（ ）友元函数可以是其他类的成员函数。

（ ）友元函数可以访问类的私有成员。

（ ）不能修改常对象的数据成员，但可以通过常对象调用普通成员函数。

（ ）常成员函数不能修改数据成员。

简答：

简述引用和指针的区别。提示：可以从是否必须初始化、能否改变、能否指向空值、sizeof()等角度分析。

const 变量在代码运行中是不可以修改的，对于定义的类，如果我们想让每个类创建出来的对象的常量值都不一样，应该怎么做呢？请创建一个类，要求同时包含一个const float和非const float，并且能用同样的方式对他们进行初始化。

**Lecture 4 继承与派生类**

**知识点**

|  |  |  |  |
| --- | --- | --- | --- |
| 在基类中的访问属性 | 继承方式 | 在派生类内部访问 | 通过派生类对象访问 |
| private | public | 不可直接访问 | 不可直接访问 |
| private | private | 不可直接访问 | 不可直接访问 |
| private | protected | 不可直接访问 | 不可直接访问 |
| public | public | public | public |
| public | private | private | 不可直接访问 |
| public | protected | protected | 不可直接访问 |
| protected | public | protected | 不可直接访问 |
| protected | private | private | 不可直接访问 |
| protected | protected | protected | 不可直接访问 |

**练习**

判断：

（ ）基类的构造函数与析构函数可以被派生类继承。

（ ）基类构造函数带有参数，则派生类必须定义构造函数。

（ ）继承的多个基类派生自同一个类时，使用 virtual 继承可以避免同名成员产生多份拷贝。

写出下列代码的输出：

*#include* <iostream>

*using* *namespace* std;

*class* A{

*public*:

A(){

cout *<<*"A() is called" *<<*endl;

}

*virtual* ~A(){

cout *<<*"~A() is called" *<<*endl;

}

};

*class* B:*public* A{

*public*:

B(){

cout *<<*"B() is called" *<<*endl;

}

~B(){

cout *<<*"~B() is called" *<<*endl;

}

};

*void* func(A a){

A*\** a1 *=* *new* A;

}

*int* main() {

cout *<<*"begin" *<<*endl;

A*\** a *=* *new* A;

func(*\**a);

B b;

}

**Lecture 5 多态性与虚函数、运算符重载**

**知识点**

*#include* <iostream>

*using* *namespace* std;

*class* Base { *// 抽象类，不能创建对象*

*public*:

Base();

*virtual* ~Base();

*virtual* *void* func1(); *// 基类的虚函数*

*virtual* *void* func2() *=*0; *// 基类的纯虚函数，无法被调用*

};

Base::Base() {

cout *<<* "Base::Base()" *<<* endl;

}

Base::~Base() {

cout *<<* "Base::~Base()" *<<* endl;

}

*void* Base::func1() {

cout *<<* "Base::func1()" *<<* endl;

}

*class* Derived: *public* Base {

*public*:

Derived(*int* \_d1, *int* \_d2);

*virtual* ~Derived();

*virtual* *void* func1(); *// 派生类的虚函数，可以没有 virtual，但一般建议加上*

*virtual* *void* func2(); *// 需要实现基类的纯虚函数*

*void* setValue(*int* \_d1); *// 函数重载*

*int* setValue(*int* \_d1, *int* \_d2); *// 函数重载*

*private*:

*int* d1;

*int* d2;

};

Derived::Derived(*int* \_d1, *int* \_d2)*:* d1(\_d1), d2(\_d2) {

cout *<<* "Derived::Derived()" *<<* endl;

}

Derived::~Derived() {

cout *<<* "Derived::~Derived()" *<<* endl;

}

*void* Derived::func1() {

cout *<<* "Derived::func1()" *<<* endl;

}

*void* Derived::func2() {

cout *<<* "Derived::func2()" *<<* endl;

}

*void* Derived::setValue(*int* \_d1) {

d1 *=* \_d1;

cout *<<* "Derived::setValue(int \_d1)" *<<* endl;

}

*int* Derived::setValue(*int* \_d1, *int* \_d2) {

d1 *=* \_d1;

d2 *=* \_d2;

cout *<<* "Derived::setValue(int \_d1, int \_d2)" *<<* endl;

*return* d2;

}

Base foo1(); *//错误，抽象类不能作为函数的返回类型*

*void* foo2(Base tmp); *//错误，抽象类不能作为函数的参数类型*

Base*&* foo3(Base*&* tmp); *//正确，可以声明抽象类的引用*

*int* main() {

Base b1; *// 错误，不能创建抽象类对象*

Base *\**ptr1; *// 正确，可以创建抽象类指针*

ptr1 *=* *new* Derived(1, 2);

*delete* ptr1; *// 会调用派生类和基类的析构函数，析构函数不加 virtual 不会调用派生类析构函数*

*return* 0;

}

*// 在类内将操作符重载为成员函数*

返回类型 operator要重载的运算符(参数列表);

*// 在类外实现成员函数*

返回类型 类名::operator要重载的运算符(参数列表) {

*// 函数体*

}

*// 在类内将操作符重载为友元函数*

*friend* 返回类型 operator要重载的运算符(参数列表);

*// 在类外实现友元函数*

返回类型 operator要重载的运算符(参数列表) {

*// 函数体*

}

**练习**

判断：

（ ）普通函数在类的内部定义，编译时认为是内联函数。

（ ）虚函数在类的内部定义，编译时认为是内联函数。

（ ）构造函数可以是虚函数。

（ ）析构函数可以是虚函数。

（ ）虚函数只能是本类的成员函数，但是可以被声明为另一个类的友元函数。

（ ）虚函数可以是静态成员函数。

（ ）不可以创建抽象类的对象，也不能创建抽象类的引用或指针。

（ ）一旦将函数声明为虚函数，即使类在重写它时没有将其声明为虚函数，它从该点之后的继承层次结构中仍然是虚函数。

（ ）派生类不重写其基类虚函数时，可直接继承其基类的虚函数。

（ ）只要将基类指针指向一个派生类对象，我们就可以用该基类指针访问派生类对象的特有成员。

（ ）派生类对象可以看成是基类对象，反过来，基类对象也可以看成是派生类对象。

（ ）试图实例化一个抽象类对象会导致运行期错误。

（ ）抽象类必须至少有一个纯虚函数。

（ ）抽象类也可以有数据成员和具体函数，包括构造函数和析构函数。

（ ）如果基类为抽象类，则不能用基类指针指向其派生类对象。

（ ）抽象类中所有虚函数都必须声明为纯虚函数。

（ ）用派生类句柄引用基类对象是安全的。

请写出两个函数，使得下面代码有注释中所给出的输出。

*int* main(){

greet("张三", 22); *// 我叫张三，今年22岁，我来自中国*

greet("李四", 21); *// 我叫李四，今年21岁，我来自中国*

greet("王五", 22, "美国"); *// 我叫王五，今年22岁，我来自美国*

greet("赵六", 23, "英国"); *// 我叫赵六，今年23岁，我来自英国*

greet("孙七", "复旦大学"); *// 我叫孙七，我来自复旦大学*

greet("周八", "上海交通大学"); *// 我叫周八，我来自上海交通大学*

}

**Lecture 7 模板**

**知识点**

*// 函数模板 1*

*template* <*class* Type…>

返回类型 函数名(参数表) {

*// 使用 Type 的函数体*

}

*// 函数模板 2*

*template* <*typename* 类型参数>

返回类型 函数名(参数表) {

*// 函数体*

}

*// 类模板 1*

*template* <*class* 类型参数>

*class* 类模板名 {

*// 类的定义体*

}**;**

*// 类模板 2*

*template* <*typename* 类型参数>

*class* 类模板名 {

*// 类的定义体*

}**;**

*// 声明模板类的对象*

类模板名<类型> 对象名;

**练习**

判断：

（ ）函数模板是一种函数，有对应的可执行代码。

（ ）一个类模板可以对应多个模板类。

需要编写一个对多维数组通用的算法（即各维的大小未定），怎样才能把实参多维数组的信息全部传递到函数中去？

什么叫函数模板？什么叫模板函数？什么叫类模板？什么叫模板类？

函数模板与同名的非模板函数重载时，调用的顺序是怎样的？